Foundations of Programming 2: Inheritance and Polymorphism

FoP 2 Teaching Team, Faculty of Computer Science, Universitas Indonesia
Correspondence: Fariz Darari (fariz@cs.ui.ac.id)

UNIVERSITAS INDONESIA
FAKULTAS ILMU KOMPUTER

Why?

Try this one, create the classes of
(incl. constructors, setters, getters, and appropriate methods):

- **Cat** with four fields:
  `name, age, can_fly, is_heterochromia`
- **Dog** with three fields:
  `name, age, can_fly`
- **Bird** with four fields:
  `name, age, can_fly, colors`

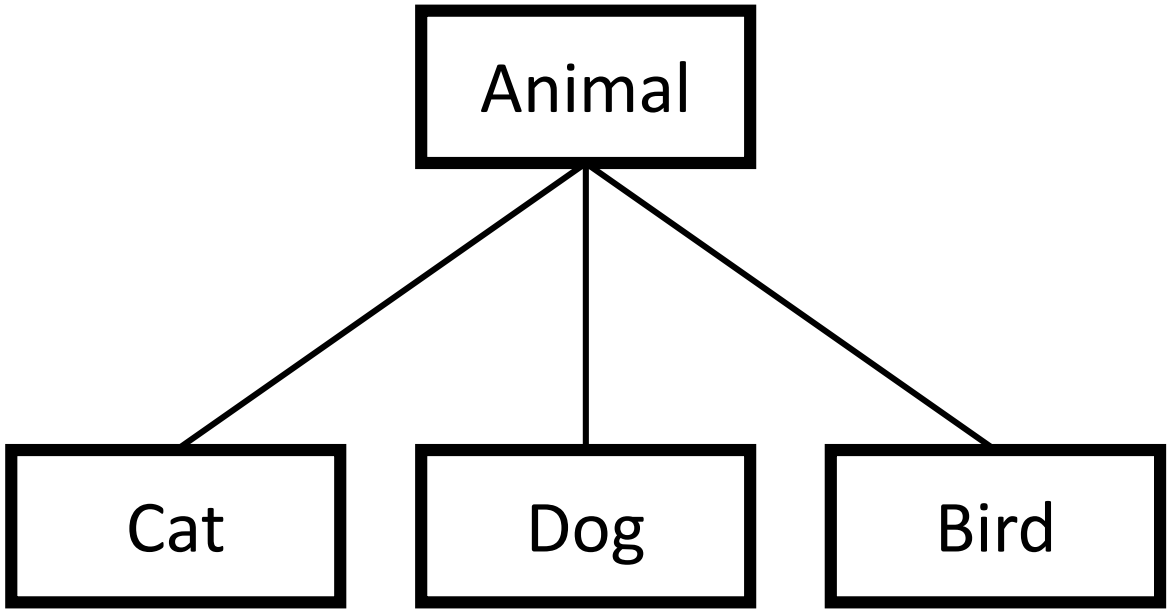
Why?

Try this one, create the classes of
(incl. constructors, setters, getters, and appropriate methods):

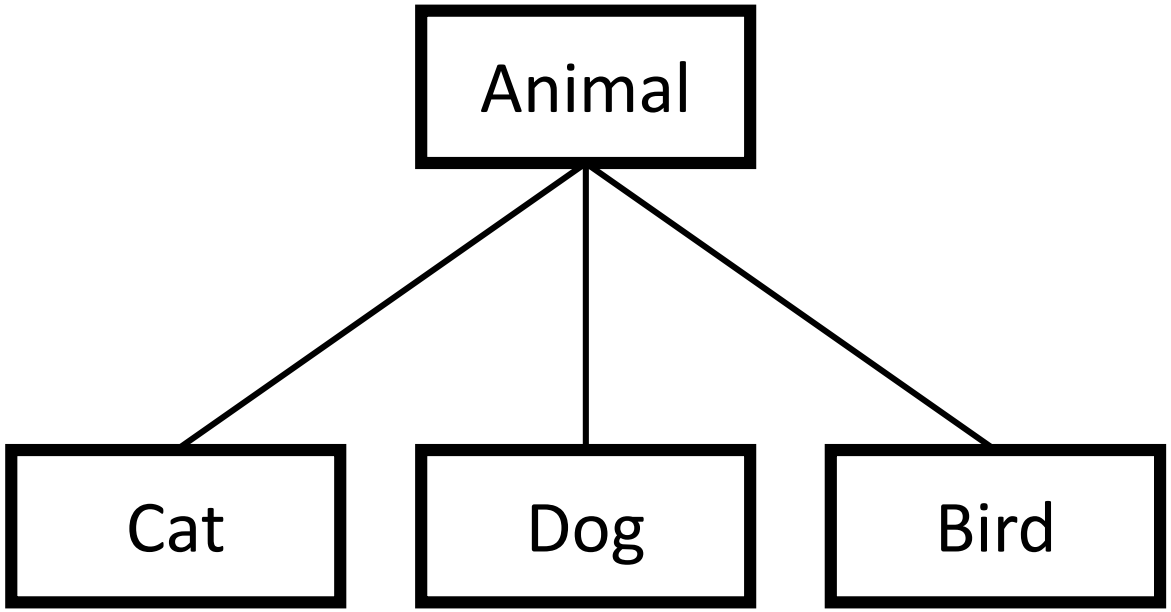
- **Cat** with four fields:
  `name, age, can_fly, is_heterochromia`
- **Dog** with three fields:
  `name, age, can_fly`
- **Bird** with four fields:
  `name, age, can_fly, colors`

What can you observe?

Inheritance



Inheritance



Which fields go to superclass and which to subclasses?

Inheritance: Animal.java

```
public class Animal {

	private String name;
	private int age;
	private boolean can_fly;

	public Animal(String name, int age, boolean can_fly) {
		this.name = name;
		this.age = age;
		this.can_fly = can_fly;
	}

// ...
```

Inheritance: Animal.java

```
// ...
	public void setName(String newName) {
		this.name = newName;
	}

	public String getName() {
		return this.name;
	}

	public void getOlder() {
		this.age++;
	}

	public int getAge() {
		return this.age;
	}
// ...
```

Inheritance: Animal.java

```
// ...
	public boolean canFly() {
		return can_fly;
	}

	public String toString() {
		return this.getName() + ", "
				+ this.getAge() + ", "
				+ this.canFly();
	}

}
```

Inheritance: Cat.java

```
public class Cat extends Animal {

	private boolean is_heterochromia;

	public Cat(String name, int age, boolean
is_heterochromia) {
		super(name, age, false);
		this.is_heterochromia = is_heterochromia;
	}

// ...
```

Inheritance: Cat.java

```
public class Cat extends Animal { // Subclass extends Superclass

	private boolean is_heterochromia;

	public Cat(String name, int age, boolean
is_heterochromia) {
		super(name, age, false);
		this.is_heterochromia = is_heterochromia;
	}

// ...
```

A subclass is a new class that **extends** an existing class; that is, it has the attributes and methods of the existing class, plus more.

Inheritance: Cat.java

```
public class Cat extends Animal {

	private boolean is_heterochromia;

	public Cat(String name, int age, boolean
is_heterochromia) {
		super(name, age, false);
		this.is_heterochromia = is_heterochromia;
	}

// ...
```

super refers to the superclass of the current class; when **super** is used like a method, it invokes the constructor of the superclass

Inheritance: Cat.java

```
// ...

    public boolean is_heterochromia() {
        return is_heterochromia;
    }

    public String toString() {
        return super.toString() + ", " + is_heterochromia();
    }

}
```

super can also be used to access the methods of the superclass

Inheritance: Dog.java

```
public class Dog extends Animal {

	public Dog(String name, int age) {
		super(name, age, false);
	}

}
```

Inheritance: Dog.java

```
public class Dog extends Animal {

	public Dog(String name, int age) {
		super(name, age, false);
	}

}
```

super refers to the superclass of the current class; when **super** is used like a method, it invokes the constructor of the superclass

Inheritance: Dog.java

```
public class Dog extends Animal {

	public Dog(String name, int age) {
		super(name, age, false);
	}

	public String toString() {
		return "Guk, guk!";
	}

}
```

A subclass can **override** the methods of its superclass

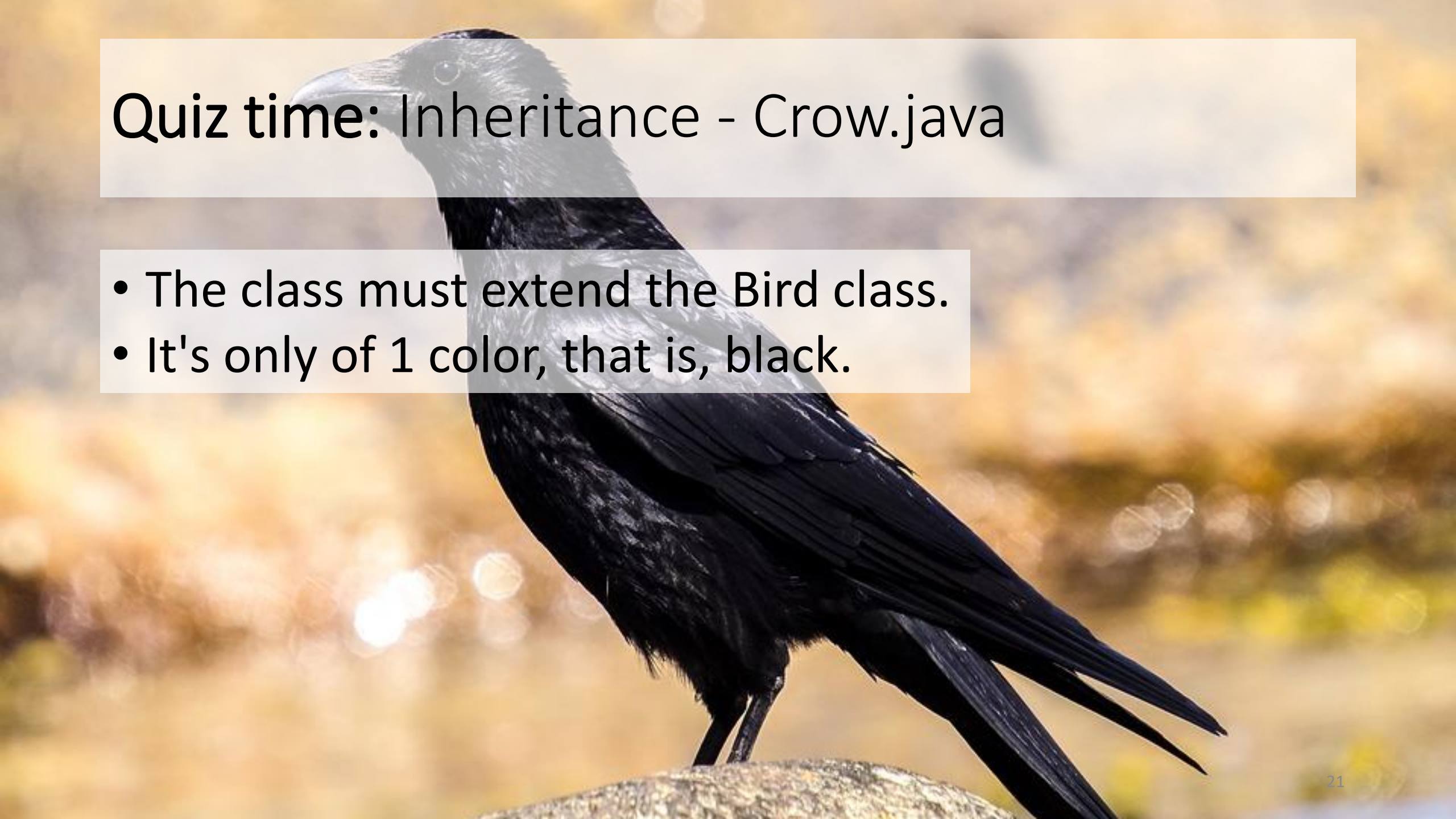
Quiz time: Inheritance - Bird.java

Quiz time: Inheritance - Bird.java

```
public class Bird extends Animal {

	private ArrayList<String> colors;

	public Bird(String name, int age, ArrayList<String> colors) {
		super(name, age, true);
		this.colors = colors;
	}

	public ArrayList<String> getColors() {
		return colors;
	}

	public String toString() {
		return super.toString() + ", " + this.getColors();
	}

}
```

Quiz time: Inheritance - Crow.java

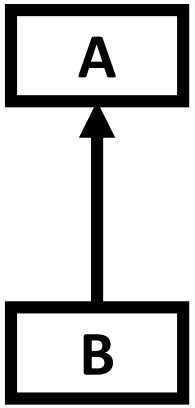
- The class must extend the Bird class.
- It's only of 1 color, that is, black.



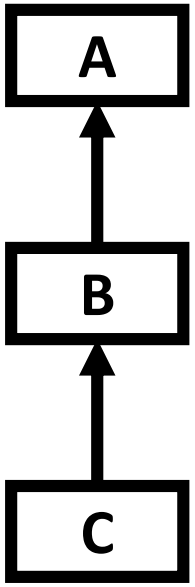
Quiz time: Inheritance - Crow.java

```
public class Crow extends Bird {

	public Crow(String name, int age) {
		super(name, age, new ArrayList<String>(Arrays.asList(new
String[]{"Black"})));
	}

}
```

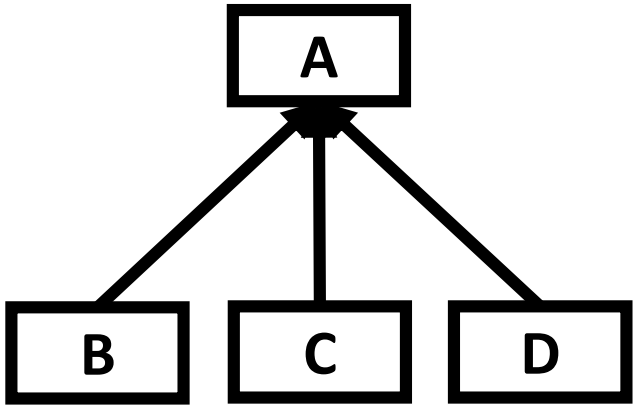
Inheritance Types



Single Inheritance



Multilevel Inheritance



Hierarchical Inheritance

More on super

- super() as a method must be the first statement in the constructor
- Even if there is no super(), it's actually called automatically

```
public class A {}
```

```
public class B extends A {
    public B() {
        // some statements
    }
}
```

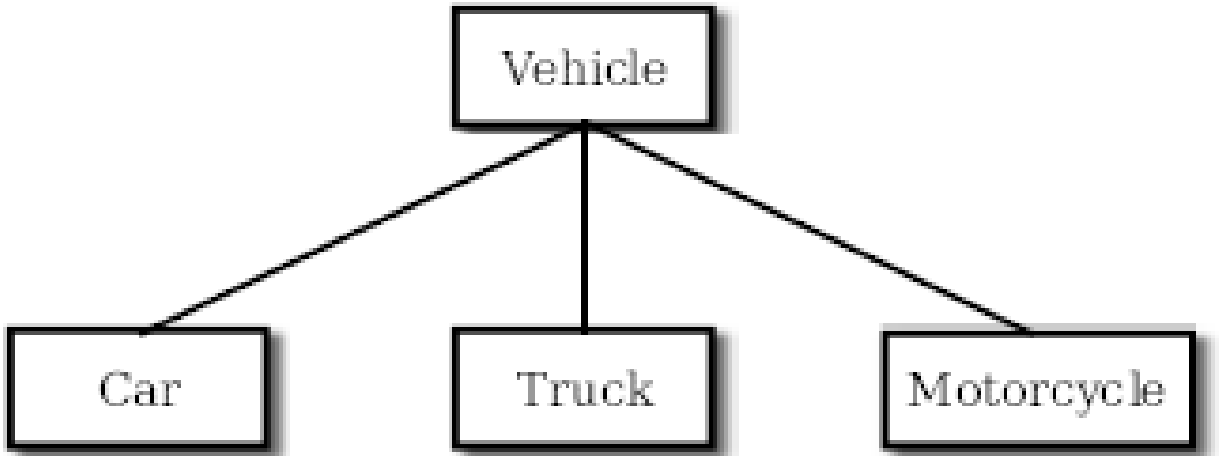
```
public class A {}
```

```
public class B extends A {
    public B() {
        super();
        // some statements
    }
}
```

Quiz time: super trap, what goes wrong?

```
public class Cockatoo extends Bird {

	public Cockatoo() {}

}
```

Quiz time: Vehicle and subclasses, specify what are the shared fields, and specific fields



What we've learned so far..

- Superclass: more generic class
- Subclass: more specific class, that inherits from a more generic class
- In Java, a class can only have at most one superclass
- What is inherited from a superclass to its subclasses:
  - Fields/attributes
  - Methods
- We can add new fields/methods in a subclass
- We can also override methods occurring in a superclass
  - Override: A method that is of the same signature (that is, same name and parameters) as that of superclass, but is implemented differently

Quiz time: The use of + here, is it overriding, or overloading?

```
System.out.println("1" + "1");
System.out.println(1 + 1);
```

Quiz time: Can you make an example of method overriding?

Quiz time: What is the root of all Java classes?

Polymorphism

Poly + morphism

- Poly = many
- Morphism = the state of having a specified shape/form

Hence, polymorphism:

The ability to assume different forms or shapes.

Recall the Animal examples, you can do this..

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
```

Recall the Animal examples, you can do this..

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
```

The object `new Crow("Crowy", 1)` has multiple forms:
Crow, Bird (Crow's superclass), Animal (Bird's superclass)

Recall the Animal examples, you can do this..

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
```

The object `new Crow("Crowy", 1)` has multiple forms: Crow, Bird (Crow's superclass), Animal (Bird's superclass)

Upcasting = Going from more specific to more general

Upcasting makes method call more simple..

.. because now instead of making a method for every class..

why not to make just one method for the superclass!

Downcasting (oops!)

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
Crow cr2 = cr1AsAnimal;
```

Downcasting

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
Crow cr2 = (Crow) cr1AsAnimal;
```

Downcasting, however..

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
Cat cr2 = (Cat) cr1AsAnimal;
```

Downcasting with instanceof

```
Crow cr1 = new Crow("Crowy", 1);
System.out.println(cr1);
Bird cr1AsBird = cr1;
Animal cr1AsAnimal = cr1;
if(cr1AsAnimal instanceof Crow) {
    Crow cr2 = (Crow) cr1AsAnimal;
    System.out.println(cr2);
}
```

Calling appropriate methods

```
Object d2 = new Dog("Helly", 3);
Object a3 = new Animal("Roar", 2, false);
System.out.println(d2);
System.out.println(a3);
```

Calling appropriate methods

```
Object d2 = new Dog("Helly", 3);
Object a3 = new Animal("Roar", 2, false);
System.out.println(d2);
System.out.println(a3);
```

Printing:
Guk, guk!
Roar, 2, false

Why upcasting? (continuation from prev examples)

```
ArrayList<Animal> animalList = new ArrayList<Animal>();
animalList.add(a1); // animal
animalList.add(c1); // cat
animalList.add(d1); // dog
animalList.add(b1); // bird
animalList.add(cr1); // crow
System.out.println("Animals:");
for(Animal an:animalList)
     System.out.println(an);
```

Quiz time: EthnicGroup

Create the class of EthnicGroup with the method goodMorning, printing out "Good morning!". The class has the following subclasses:

- JavaneseEthnicGroup, with the method goodMorning -> "Sugeng enjing!"
- SundaneseEthnicGroup, with the method goodMorning -> "Wilujeng enjing!"
- BatakneseEthnicGroup, with the method goodMorning -> "Horas!"

Then, create a main class that instantiates all the ethnic groups (incl. the default group), where the instantiations call goodMorning().

What can you observe?

Inspired by:

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
Liang. Introduction to Java Programming. Tenth Edition. Pearson 2015.
Think Java book by Allen Downey and Chris Mayfield.
Eck. Introduction to Programming Using Java. 2014.