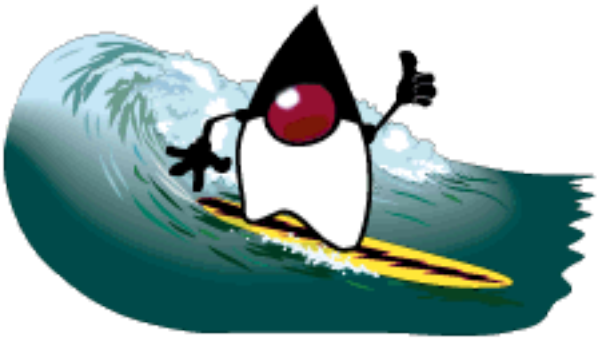
T5 - Java Seminar

T-JAV-500

Day 04

Inheritance



3.2

Day 04

language: Java

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

Let's delve again deeper into OOP.
Today, you will keep using all of the previous' days concepts, and you will also discover a few more concepts:

- enum
- static keyword with methods (also called "static method")
- inheritance
- protected visibility

Here is a quick example of inheritance.

Let's imagine an `Apple` class inheriting from a `Fruit` class. This `Apple` class will have its own method, like `peel` and will also possess all of the methods and attributes that weren't private in its parent class (Fruit); for example, an `eat` method.
Inheritance trees are limitless. In this example we could have a `GoldenApple` class inheriting from the `Apple` class. It would inherit all of the properties that weren't private in the `Apple` class, which means that it would obviously also inherit the `Fruit` class characteristics.

Inheritance is probably the most important notion in OOP.

Unless specified otherwise, all messages must be followed by a newline.

Unless specified otherwise, the getter and setter name will always follow this format: **getAttribute** or **setAttribute**.
If you attribute's name is **someWeirdNameAttribute**, its getter will be **getSomeWeirdNameAttribute** (FYI, this name convention is known as "CamelCase").

EXERCISE 01

File to hand in: ./Animal.java

Create a new `Animal` class that has a protected enum `Type` with 3 possible values:
MAMMAL, **FISH**, **BIRD**.

These variables will be used to pass our Animal's type to the protected constructor, which takes 3 mandatory parameters:

1. the name of our animal,
2. its number of legs,
3. its type, among the three previously-created attributes (MAMMAL, FISH, BIRD).

These 3 parameters must be stored inside new attributes, respectively named "name", "legs" and "type". Create getters for each of these attributes.

Even though accidents can happen, we are generous Gods (well, at least, I am!) and will consider that our animals will never lose a leg after creation (and also, because I love cats).
That's why there is no setter for this attribute.
For the other attributes, you will need to think about it by yourself because there are some logical reasons for it.

Be careful, the getter for "type" doesn't return the enum value directly.
You will need to make it work like in this example (and just guess for the other cases).

Last, but not least, during its creation our Animal must say:

```
My name is [name] and I am a [type]!
```

where [name] is the name of the Animal and [type], its type.

The constructor is `protected` and not `public`, thus there may be cases when you can't instantiate the object directly. Read the documentation for more informations.

```
public class Example {
    public static void main(String[] args) {
    Animal isidore = new Animal("Isidore", 4, Animal.Type.MAMMAL);

    System.out.println(isidore.getName() + " has " + isidore.getLegs()
                + " legs and is a " + isidore.getType() + ".");
    }
}
```

```
Terminal
~/T-JAV-500> java Example
My name is Isidore and I am a mammal!
Isidore has 4 legs and is a mammal.
```

EXERCISE 02

File to hand in: ./Animal.java

Implement a private static field `numberOfAnimals` and it's getter **getNumberOfAnimals**, that returns the number of `Animal` instances and displays it in a sentence like the following:

```
There is(are) currently [x] animal(s) in our world.
```

where [x] is the number of currently alive animals.

The method to handle this is up to you.

You must also implement 3 variants of this for the following private static fields: **numberOfMammals**, **numberOfFish** and **numberOfBirds**.
They respectively return the number of mammals, fish and birds that are alive in the world and display the following message:

```
There is(are) currently [x] [type](s) in our world.
```

where [x] is the number of animals of this type that are currently alive, and [type] is the type of animal of the function being called.
An example would be "There are currently 3 mammals in our world.".

In this exercise, use plural when necessary.
The plural of "fish" is "fish".

Animals never die.

EXERCISE 03

Files to hand in: ./Cat.java
./Animal.java

Create a new "Cat" class that inherits from "Animal" and has a private "color" attribute, with a getter only.

When "Cat" is created, display:

```
[name]: MEEEOOWWWW
```

where [name] is…the name of the cat!

The name of the cat should always be specified as the first parameter of its constructor.
However, the color can be specified as its second parameter, but is not mandatory.
Its number of legs must be set to 4 and its type to MAMMAL by default.

If no color has been specified during creation, the default color will be grey.

In Java a constructor can call a constructor from the upper class. Use it!

Add a public **meow** method to your "Cat" class.
This method does not take any parameters, and when calling it, it displays the following message:

```
[name] the [color] kitty is meowing.
```

where [name] is the name of the cat and [color] its color.

```
public class Example {
    public static void main(String[] args) {
        Cat isidore = new Cat("Isidore", "orange");

        System.out.println(isidore.getName() + " has " + isidore.getLegs()
                + " legs and is a " + isidore.getType() + ".");
        isidore.meow();
    }
}
```

Terminal

```
~/T-JAV-500> java Example
My name is Isidore and I am a mammal!
Isidore: MEEEOOWWWW
Isidore has 4 legs and is a mammal.
Isidore the orange kitty is meowing.
```

EXERCISE 04

Files to hand in: ./Cat.java
./Animal.java
./Shark.java
./Canary.java

Create “Shark” and “Canary” class, which both inherit from “Animal”.

When a “Shark” is created, display:

```
A KILLER IS BORN!
```

The “Shark” class must also have a private attribute named “frenzy”.

When the class is constructed, it receives its name as parameter.
Its number of legs should be set to 0 and its type to “FISH”.
Its “frenzy” attribute should be on “false” by default.

Also add a **smellBlood** method to it.
It takes a Boolean as parameter and returns nothing.
This method changes the value of “frenzy” to the value passed as parameter.

Finally, you must add a **status** method that displays one of the two following messages, depending on frenzy:

```
[name] is smelling blood and wants to kill.
```

if “frenzy” is true, and

```
[name] is swimming peacefully.
```

if “frenzy” is false, where [name] is the shark’s name.

Your “Canary” class must have a private “eggs” attribute, which indicates how many eggs it has laid in its life.
During initialization, “Canary” only takes one parameter: its name.
“legs” is initialized to 2, “type” to BIRD, and “eggs” to 0.

During its creation, it must say:

```
Yellow and smart? Here I am!
```

You must add a **getEggsCount** method that returns the number of eggs laid by the Canary.
And a **layEgg** method that increases the number of eggs laid by 1.

```
    public class Example {
        public static void main(String[] args) {
            Canary titi = new Canary("Titi");
            Shark willy = new Shark("Willy"); //Yes Willy is a shark here !

            willy.status();
            willy.smellBlood(true);
            willy.status();

            titi.layEgg();
            System.out.println(titi.getEggsCount());
     }
}
```

Terminal

```
~/T-JAV-500> java Example
My name is Titi and I am a bird!
Yellow and smart? Here I am!
My name is Willy and I am a fish!
A KILLER IS BORN!
Willy is swimming peacefully.
Willy is smelling blood and wants to kill.
1
```

EXERCISE 05

Files to hand in: ./Cat.java
./Animal.java
./Shark.java
./Canary.java

Willy, our dear shark, needs to eat…
Add a public **canEat** method to your `Shark` class.
It takes an `Animal` as parameter and returns a boolean indicating wether or not the `Shark` can it the `Animal`.
Add another public method: **eat**, also taking an `Animal` as parameter.
When the method is called, display:

```
[Shark's name] ate a [Animal's type] named [Animal's name].
```

where [Shark's name] is the name of the shark,
[Animal's type] is the type of the animal being eaten (mammal, fish, bird)
[Animal's name] is the name of the animal being eaten.

If the parameter can't be eaten, your shark must say:

```
[name]: It's not worth my time.
```

where [name] is the name of the shark.

If the "Shark"'s "frenzy" attribute is set to true and our shark has successfully eaten, it must be set to false after calling this method.

Your shark cannot eat itself.

EXERCISE 06

Files to hand in: ./Cat.java
./Animal.java
./Shark.java
./Canary.java
./BlueShark.java
./GreatWhite.java

For this exercise, you need to create two new classes: “BlueShark” and “GreatWhite”, which both inherit from the “Shark” class.
They both take their name during construction as a mandatory argument.

They are almost identical to the original "Shark' class, except that they are picky eaters.

“BlueShark” refuses to eat anything but “Fish”. For that, only the `canEat` should be modified.
“GreatWhite” refuses to eat “Canary”. Even worse, if you try to feed a “Canary” to a “GreatWhite”, it will only say:

```
[name]: Next time you try to give me that to eat, I'll eat you instead.
```

where [name] is the name of the GreatWhite.

Except for those minor differences, their “eat” function must do the exact same thing as for the “Shark” Class.

… Oh, I almost forgot!
If a GreatWhite eats another Shark, it must display (after it's done eating):

```
[name]: The best meal one could wish for.
```

where [name] is the name of the GreatWhite.

Cleverly use the `Override` annotation and the `instanceof` keyword.