Fondamenti della Programmazione: Metodi Evoluti

Prof. Enrico Nardelli

Esercitazione 6

Remember the Acrobat game

- First there was an acrobat object.
  - When asked to **Clap**, it will be given a number and it has clap its hands that many times.
  - When asked to **Twirl**, it will be given a number and it has to turn completely around that many times.
  - When asked for **Count**, it has to announce how many actions it has performed. This is the sum of the numbers that have been given to date.
- Then a copycat was assigned to an acrobat, able to do the same action as the acrobat.
- Whenever the acrobat is asked to do something it does it and ask the same to its copycat
- Whenever the acrobat is asked for **Count** it provides the answer given by its copycat

There was an *ACROBAT* – without copycat

```
class
    ACROBAT
feature
    clap (n: INTEGER)
            -- Clap `n' times and adjust `count'.
        require   n>0
        do
        -- to be completed
        ensure    count = old count + n
        end
    twirl (n: INTEGER)
            -- Twirl `n' times and adjust `count'.
        require   n>0
        do
        -- to be completed
        ensure    count = old count + n
        end
    count: INTEGER
            -- Total # of times clapped or twirled.
end
```

There was an *ACROBAT* - with copycat (1)

```
class
    ACROBAT
feature
    buddy: COPYCAT
    pair (p: COPYCAT)
            -- Remember `p' being the copycat.
            do
        -- to be completed
        ensure    buddy = p
        end
```

There was an *ACROBAT* with copycat (2)

```
    clap (n: INTEGER)
            -- Clap `n' times and forward to copycat.
        do
        -- to be completed
        ensure   count = old count + n
        end

    twirl (n: INTEGER)
            -- Twirl `n' times and forward to copycat.
        do
        -- to be completed
        ensure   count = old count + n
        end

    count: INTEGER
            -- Ask copycat and return his answer.
        do
        -- to be completed
        end

end
```

There was a *COPYCAT*

```
class
    COPYCAT
feature
    clap (n: INTEGER)
            -- Clap `n' times and adjust `count'.
        require   n>0
        do
        -- to be completed
        ensure    count = old count + n
        end
    twirl (n: INTEGER)
            -- Twirl `n' times and adjust `count'.
        require   n>0
        do
        -- to be completed
        ensure    count = old count + n
        end
    count: INTEGER
            -- Total # of times clapped or twirled.
end
```

Hands-On

But now we can use inheritance to model the COPYCAT

What do we model at super-class level?

What at sub-class level?

Different choices are posible

You are an *ACROBAT* – generic version

```
class
    ACROBAT
feature
    clap (n: INTEGER)
            -- Clap `n' times and adjust `count'.
        require  n>0
        do
        -- to be completed
        ensure   count = old count + n
        end
    twirl (n: INTEGER)
            -- Twirl `n' times and adjust `count'.
        require  n>0
        do
        -- to be completed
        ensure   count = old count + n
        end
    count: INTEGER
            -- Total # of times clapped or twirled.
end
```

You are an *ACROBAT_WITH_COPYCAT* (1)

```
class
    ACROBAT_WITH_COPYCAT
inherit
    ACROBAT
        redefine
            twirl, clap, count
        end
create
    pair
feature
    buddy: ACROBAT
    pair (p: ACROBAT)
            -- Remember `p' being the copycat.
        do
        -- to be completed
        ensure   buddy = p
        end
```

You are an *ACROBAT_WITH_COPYCAT* (2)


```
    clap (n: INTEGER)
            -- Clap `n' times and forward to copycat.
        do
        -- to be completed
        end

    twirl (n: INTEGER)
            -- Twirl `n' times and forward to copycat.
        do
        -- to be completed
        end

    count: INTEGER
            -- Ask copycat and return his answer.
        do
        -- to be completed
        end

end
```

You are an *ACROBAT_WITH_COPYCAT* – implementation

Open EiffelStudio,

copy-paste the code,

and complete it !

Cannot redefine an attribute in descendants!

- Not allowed by the language definition.
- It might break code in ancestors where a value might be assigned to the attribute
- Performance degradation: replacing a simple memory access with a function call
- It might make things slower for incremental compiler and slow down dynamic access to entities.
- Choice might change in the future (even if unlikely)

- SOLUTION
- Hide the attribute in the ancestor
- Expose a function which is inherited and redefined

You are an *ACROBAT* – new solution

```
class
    ACROBAT
feature {NONE}
    internal_count: INTEGER
            -- Store value of a private counter.
feature
    count: INTEGER
            -- Total # of times clapped or twirled.
        do
            Result := internal_count
        end

    clap (n: INTEGER)
            -- Clap `n' times and adjust `count'.
        require    n>0
        do
            internal_count := internal_count + n
        ensure     internal_count = old internal_count + n
        end

    twirl (n: INTEGER)
            -- Twirl `n' times and adjust `count'.
        require    n>0
        do
            internal_count := internal_count + n
        ensure     internal_count = old internal_count + n
        end
```

You are an *ACROBAT_WITH_COPYCAT* – implementation

Open EiffelStudio,

copy-paste the code,

and complete it !

You are an *ACROBAT_WITH_COPYCAT* – new sol. (1)

```
class
    ACROBAT_WITH_COPYCAT
inherit
    ACROBAT
        redefine
            twirl, clap, count
        end
create
    pair
feature
    count: INTEGER
            -- Ask copycat and return his answer.
        do
            Result := buddy.count
        end
    buddy : ACROBAT
    pair (p: ACROBAT)
            -- Remember `p' being the copycat.
        do
            buddy := p
        ensure      buddy = p
        end
```

You are an *ACROBAT_WITH_COPYCAT* – new sol. (2)

```
    clap (n: INTEGER)
            -- Clap `n' times and forward to copycat.
        do
            Precursor(n)
            buddy.clap(n)
        end
    twirl (n: INTEGER)
            -- Twirl `n' times and forward to copycat.
        do
            Precursor(n)
            buddy.twirl(n)
        end
end
```

I am the root object (version 2)

```
prepare_and_play
        local
            mario, luigi, piero: ACROBAT
            maria, luigi, piera: ACROBAT_WITH_COPYCAT


        do
            create mario
            create luigi
            create piero
            create maria.pair (mario)
            create luigia.pair (luigi)
            create piero.pair (piera)
            maria.twirl (2)
            luigi.clap (7)
            piera.clap (6)
        end
```

N.B. Allow objects to give feedback to what happens to them by printing it.
For example: add an attribute name: STRING and print it when acrobats report

You are an author

- When you are asked to **Clap**, you will be given a number. Clap your hands that many times. Say "Thank You." Then take a bow (as dramatically as you like).
- When you are asked to **Twirl**, you will be given a number. Turn completely around that many times. Say "Thank You." Then take a bow (as dramatically as you like).
- When you are asked for **Count**, announce how many actions you have performed. This is the sum of the numbers you have been given to date.

You are an *AUTHOR* – initial

```
class
    AUTHOR

inherit
    ACROBAT
        redefine
            clap, twirl
        end

feature
    clap (n: INTEGER)
        do
            -- Clap `n' times say thanks and bow.
        end

    twirl (n: INTEGER)
        do
            -- Twirl `n' times say thanks and bow.
        end
end
```

You are an *AUTHOR* – invariants

```
class
    AUTHOR

inherit
    ACROBAT
        redefine
            clap, twirl
        end

feature
    clap (n: INTEGER)
            -- Clap `n' times say thanks and bow.
        do
        -- to be completed.
        end

    twirl (n: INTEGER)
            -- Twirl `n' times say thanks and bow.
        do
        -- to be completed.
        end
end
```

You are an *AUTHOR* – implementation

Open EiffelStudio,

copy-paste the code,

and complete it !

I am the root object (version 3)

```
prepare_and_play
        local
            mario, luigi, piero: ACROBAT
            maria, luigi, piera: ACROBAT_WITH_COPYCAT
            dante: AUTHOR

        do
            create mario
            create luigi
            create piero
            create maria.pair (mario)
            create luigia.pair (luigi)
            create piero.pair (piera)
            create dante
            maria.twirl (2)
            luigi.clap (7)
            piera.clap (6)
            dante.clap (4)
        end
```