EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W8

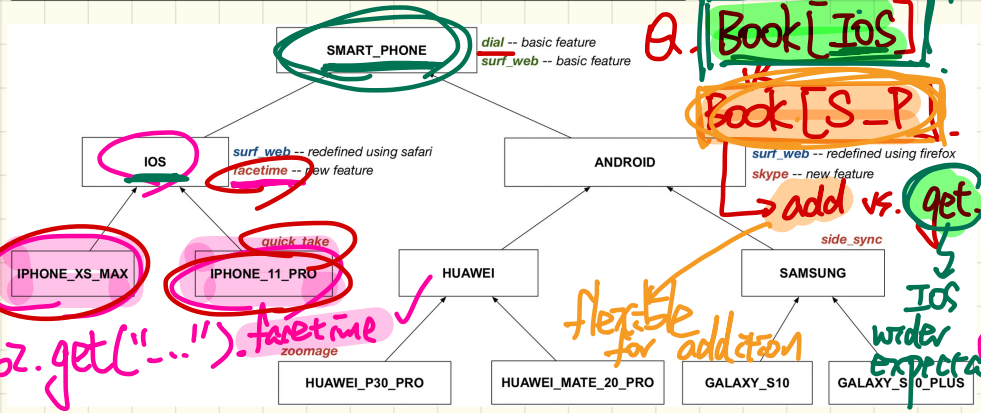
Monday, November 9

In lecture 8a part 2 we started by the motivation example: an array with a mixture of polymorphic types (STRING, DATE) and the problem is that how to have single choice for accessing and casting each element.

How does generic solve this problem?

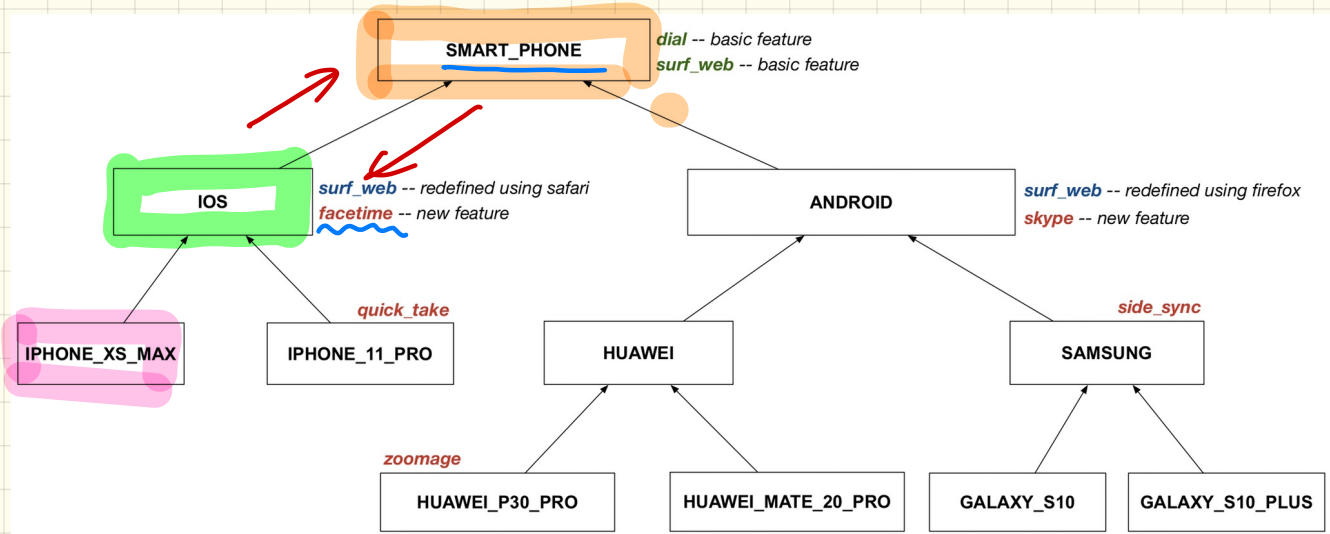
I mean using Generic changes the problem statement. Giving us an array of G not mixture of G, H, ...

General vs. Generic Book




```
class BOOK
  names: ARRAY[STRING]
  records: ARRAY[ANY]
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

```
class BOOK[X]
  names: ARRAY[STRING]
  records: ARRAY[X]
  -- Create an empty book
  make do ... end
  /* Add a name-record pair to the book */
  add (name: STRING; record: X) do ... end
  /* Return the record associated with a given name */
  get (name: STRING): X do ... end
end
```



mine : IOS

create {IPHONE_XS_MAX} mine. make.

check attached {SMART_PHONE} mine as to_lend then

end → to_lend. facetime ✗.

```
class
    B
inherit
    G
    redefine fa end
feature
    fa: STRING
        do
            Result := "B.fa"
        end
    fb: STRING
        do
            Result := "B.fb"
        end
end
```

```
class
    D
inherit
    A
    redefine fa end
feature
    fa: STRING
        do
            Result := "D.fa"
        end
    fd: STRING
        do
            Result := "D.fd"
        end
end
```

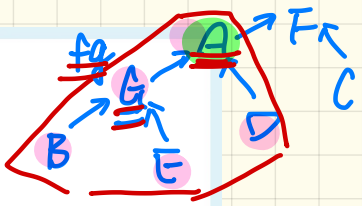
```
class
    G
inherit
    A
    redefine fa end
feature
    fa: STRING
        do
            Result := "G.fa"
        end
    fg: STRING
        do
            Result := "G.fg"
        end
end
```

```
class
    F
feature
    ff: STRING
        do
            Result := "F.ff"
        end
end
```

```
class
    C
inherit
    F
feature
    fc: STRING
        do
            Result := "C.fc"
        end
end
```

```
class
    E
inherit
    G
feature
    fe: STRING
        do
            Result := "E.fe"
        end
end
```

```
class
    A
inherit
    F
feature
    fa: STRING
        do
            Result := "A.fa"
        end
end
```



obj → G

obj : A

create {G} obj.make

obj.fg.X

∵ obj's ST ⇒ A which doesn't include EXP of fg.

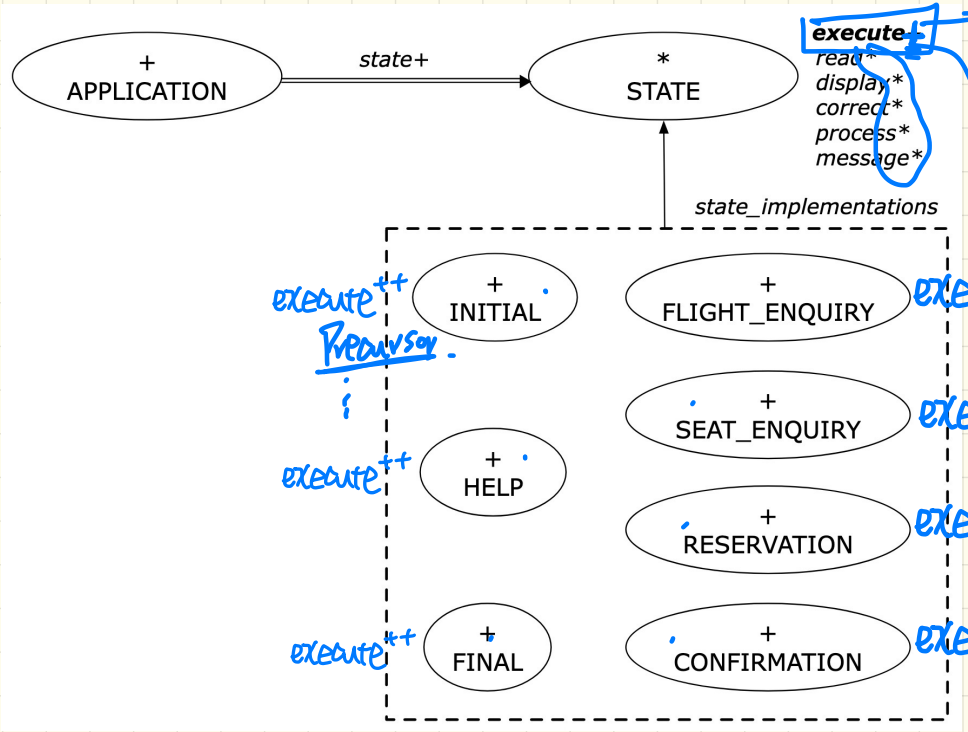
obj: A

create {T} obj.make

any T that can fulfill expectation on A should be valid

⇒ A, B, G, E, D (descendants of A).

In the above object creation, what can be a valid dynamic type **T**?



execute*

execute++

execute++ Recursion

execute++

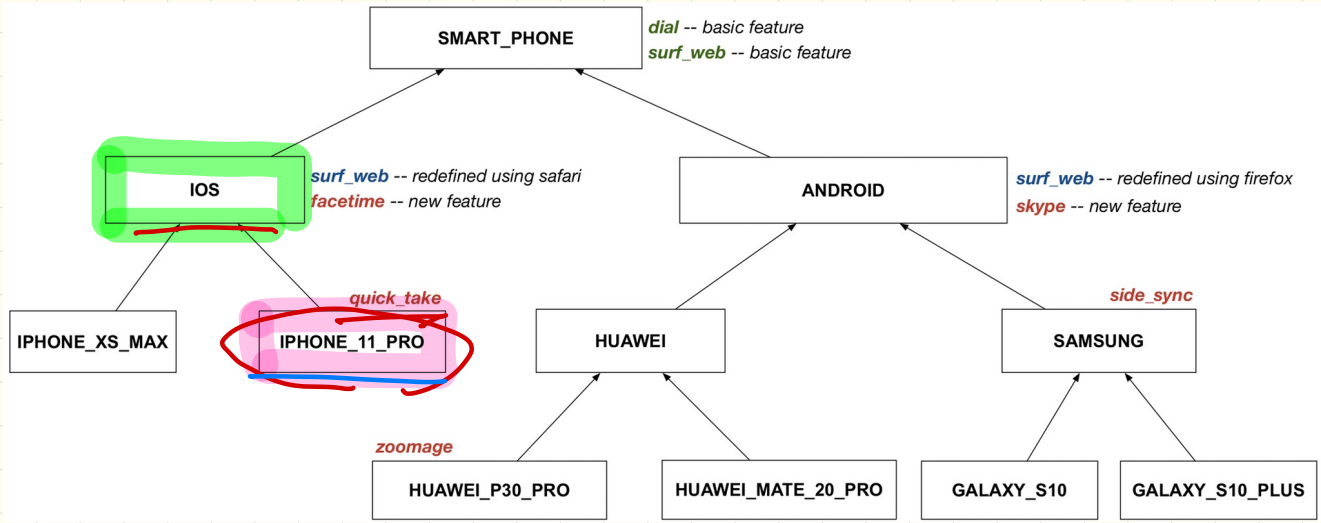
execute++

execute++

execute++

execute++

⇒ execute from STATE is not really a common template for all state dependant classes to use.



mine : IOS

Create { IPHONE_11_PRO } mine.make

IOS mine

IP_11_Pro ip

IP_11_Pro

mine . quick_take ✗ → check attached { IPHONE_11_PRO } mine as ip then

end ip. quick_take ✓