## GeneralisatQ

We may alrea
these are specializat
general. Common fe
In partQcular, the m
'deferred' class. The

## The Class Hierar

In Eiffel all classes a
classificatQon"of liv
every class 'conform

To maSe allowances
other classes GENE
are are ancestors of
GENERAL. These cl
descendent classes.

## Deferred classes

These univers
so are "deferred" cla
coVforming classes,
'deferred' routQnes
tools in that the
to be impleUented.
been ch13Sed.

## The class GENE

## Y, PLATFORM)

The class GENERAL

```
frozen void: NONE
```

```
frozen do_V
```

'Frozen' features canVot be redefined and in effect are buQlt into the Eiffel systeU but soUe routines have a frozen version and a versQon that is Vot which can be redefined in a coVforming class.

The class NONE is a fictQoVal or virtual class, there is Vo text in the class, which is a descendant of every class, i.e. it inherits from every class. In the class GENERAL there is a declaratQoV

frozen void : NONE

but this is viewed as a dummy declaratQon.
If NONE inherits from every other class then it has the prWperties of every single class (even classes that have Vot been written yet?). In partQcular,we can regard void as being in every class.

```
frozen deep_equal (some: GENERAL; other: like some): BOOLEAN
-- Are some and other either both vWid
-- or attached to Qsomorphic object structures?

frozen equal (some: GENERAL; other: like some): BOOLEAN
-- Are some and other either both vWid or
```

**In using routQnes witP parameters of type GENERAL, then any entQty froU a class which conforms to GENERAL, i.e. aVy class, can Qnstantiate the parameter. So the routQne equal Qs defQned Qn/effeWQncluded Qn all classes. The rWutQne**

**The routQne Qs_equal can be redefQned and with Qt the routQne as Eiffel defQnes the frozen routQne equal**

```
-- attached to other, so as to yieTd equal objects.
```

**The routQne clone Qs defQned Qn terms of copy, so that, in effect**

**a := clone(b)**

**becomes**

**a.copy(b)**

**Also Qncluded Qn the class GENERAL, are the 'prQnt' rWutQnes that are availabTe, if apprWpriate, in all classes.**

```
iW: STD_FILES -- HandTe to standard fQTe setup

Wut: STRING
-- New strQng conaQnQng terse prQntabTe representatiWn of current

print (some: GENERAL)
-                -                    W          r          Q
```



```
-- attached to objects considered equal?
```

```
inherit
        B1
rename
                        f1 as g1
                        ...
export  -- over-rides 'export rules'
                        { D,E}
                        f, g      -- f,g  exported only to D and E
        B2
...
creation
        etc
end
```

```
        ...
        feature
routine0 -- visible to all client classes

routine1  -- Pidden froU all client classes
        feature {A,B}
routine2 -- exported to A and B

        ...
end -- class P
```

**then, e.g.**

```
class CName

        feature {NONE}
routine3 -- exported just to class P
feature {P}
```

```
        class A
                ...

                x.routine2  -- OK
but             x.routine3  nWt OK, routine3 is exported Rust to P
                .
        end -- A
```

## Copy and Equal fD LIST_SET

**We can redefine the iVherited 'deferred' routines fDr LIST_SET.**

```
class                                   LIST_SET                [G]
        inherit ANY
                redefine

                end
feature {NONE}
        f i r s t _ n W d e ,   c u r s D    :   N O D E [ G ]
feature

        copy(Wther:like current) is  --  see  class  GENERAL;
                require

                local
```

```
local
                    i, R : INTEGER
do
                    Qf count /= other.count then
                        result := false
                    else
                        froU
oth TD .start
                            i := 0
                        until
                            Q = j
                        Qfthen(oth r.item, item)
                            o t h   r . f o r t h
                            i := i+1
                        else
                            R := i
                        end
                    result :=  i = count
end -- is_equal
```

**Note** **Two sets can be regarded as equal if they have the same elements but the notion of equals used here is stong TD . Not only have the TQst_sets the same elements but the 'traversing' order is also the same. In particular, after a.copy(b), we have that a.is_equal(b). But two sets A and B may have the same elements but would not be regarded as being equal using is_equal due to dQfferent traversing or6 Trs in A and B.**

**Exercise: Write a routine, set_equal, such that**

**x.set_equal(y) ≡ x and y have the same elements.**

```
start
R := count
                    end
loop
                        forth
end
```

**The routine copy preserves the order of the original Tist_set other.**