CAT calls
(Changed Availability or Type)

these slides contain advanced material and are optional
Changed Availability or Type (CAT)

- Changed Availability (export status)
  - Lifting restrictions
  - Tighten restrictions
- Changed Type (argument or result type)
  - Covariant redefinition (changing to a narrower type)
  - Contravariant redefinition (changing to a wider type)
Why change availability or type?

- Greater expressiveness when modeling

```
BIRD

OSTRICH

fly

CUSTOMER

drink: BEVERAGE
serve (b: BEVERAGE)
do drink := b end

MINOR

SOFTDRINK

BEVERAGE

serve (b: SOFTDRINK)
do drink := b end

ALCOHOL
```
Changed availability

- Changed Availability (export status)
  - Lifting restrictions
    ```
    class B
    Inherit A
    export {ANY} f end
    end
    ```
  - Tighten restrictions
    ```
    class C
    Inherit A
    export {NONE} f end
    end
    ```
class X
feature
make
  local
    a: A
  do
    a := create {B}
    a.f
    a := create {C}
    a.f
  end
end

class A
feature {X}
  f
end

class B
inherit A
  export {ANY} f end
end

class C
inherit A
  export {NONE} f end
end

{B}.f is available to all classes, including X

{C}.f should not be available to X
Solution to availability problem

• When inheriting **conformingly**, restricting export status is **not allowed** anymore (Eiffel standard 2006)
  – EiffelStudio still allows it for backwards compatibility

• When inheriting **non-conformingly**, restricting export status is **allowed**

```eiffel
class A
  feature {X}
    f
  end
Inherit {NONE}
end

class C
  export {NONE} f end
end

local
  a: A
  do
    a := create {C}
    a.f
  end
end
```

Invalid assignment.

C does not conform to A
class A
feature
  f (a: ANY): ANY
    deferred
end
end

class B
inherit A redefine f end
feature
  f (a: STRING): ANY
    do
      a.to_upper
      create Result
    end
end

class C
inherit A redefine f end
feature
  f (a: ANY): STRING
    do
      a.do_nothing
      create Result.
      make_from_string (a.out)
    end
end

class X
feature
  make
    local
      a: A
    do
      a := create {B}
      a.f (1).do_nothing
      a := create {C}
      a.f (1).do_nothing
    end
end

Covariant redefinition of argument type: Call to invalid feature.

Covariant redefinition of result type: Call Ok.
Changed type: contravariant

Contravariant redefinition of result type: Call to invalid feature.

Contravariant redefinition of argument type: Call Ok.
ANY.is_equal

• Routine \{ANY\}.is_equal has anchored argument:

```plaintext
is_equal (other: like Current): BOOLEAN
   do ... end
```

• This is a covariant redefinition in every type
• You should never use is_equal directly
• Use the \sim\-operator, which checks the dynamic type of both entities before calling is_equal

```plaintext
f (a, b: ANY): BOOLEAN
   do
      Result := a.is_equal (b)  # could be invalid at runtime
      Result := a \sim b  # always safe
   end
```
Changed type: summary

• It is **safe** to:
  – Change result type covariantly
  – Change argument type contravariantly

• It is **unsafe** to
  – Change result type contravariantly
  – Change argument type covariantly

• Eiffel allows
  – Covariant change of result type (safe)
  – Covariant change of argument type (unsafe)
class PARENT
    feature {X}
        f (a: B): B
    end

class CHILD inherit PARENT
    feature {NONE}
        f (a: C): A
    end

class PARENT
    feature
        f (a: ANY): ANY
        require
            type_of (Caller).conforms_to ({X})
            type_of (a).conforms_to ({B})
        ensure
            type_of (Result).conforms_to ({B})
    end

child inherit PARENT
    feature
        f (a: ANY): ANY
        require
            type_of (Caller).conforms_to ({NONE})
            type_of (a).conforms_to ({C})
        ensure
            type_of (Result).conforms_to ({A})
    end

Stronger precondition

Weaker postcondition
class APPLICATION

feature make

  local
    any_list: LIST [ANY]
    string_list: LIST [STRING]
    integer_list: LIST [INTEGER]

  do
    string_list := any_list  ×
    string_list := integer_list  ×
    integer_list := any_list  ×
    integer_list := string_list  ×
    any_list := string_list  ✓
    any_list := integer_list  ✓

  end

end
class LIST [G]
feature
  put (a: G) do end
  first: G
end

interface class LIST [ANY]
feature
  put (a: ANY)
  first: ANY
end

interface class LIST [STRING]
feature
  put (a: STRING)
  first: STRING
end

LIST [STRING] conforms to LIST [ANY], thus the changed type in the argument and result are like a covariant redefinition.
Problems with generics

```plaintext
class APPLICATION
feature make
  local
    any_list: LIST [ANY]
    string_list: LIST [STRING]
  do
    create string_list.make
    any_list := string_list
    any_list.put (1)
    string_list.first.to_upper
  end
end
```

Wrong element type!
Different solutions for generics

• **Novariant conformance**
  – No conformance between generics of different type
  – Used by C#

• **Usage-site variance**
  – Specify conformance for each generic derivation
  – Used by Java

• **Definition-site variance**
  – Specify conformance for each generic class
  – Implemented by CLR
Possible solution: Type intervals

• All types have an upper and lower bound

  \[ c : \text{CUSTOMER..MINOR} \]

• Special abbreviations:

  \[ \begin{align*}
  c1 & : \text{CUSTOMER} \\
  c2 & : \text{frozen CUSTOMER}
  \end{align*} \quad \rightarrow \quad
  \begin{align*}
  c1 & : \text{CUSTOMER..NONE} \\
  c2 & : \text{CUSTOMER.. CUSTOMER}
  \end{align*} \]

• Calls on any entity have to be valid for any possible type in the interval

  \[ \begin{align*}
  c : \text{CUSTOMER..CUSTOMER} \\
  c\.serve (vodka)
  \end{align*} \quad \rightarrow \quad
  \begin{align*}
  c : \text{CUSTOMER..MINOR} \\
  c\.serve (vodka)
  \end{align*} \]

  Valid

  Invalid, since invalid for MINOR
Possible solution: usage-site variance

- Conformance of different generic instantiations require **variant** mark

```
justin: MINOR
bv: LIST [CUSTOMER]
vbus: LIST [variant CUSTOMER]
school_bus: LIST [MINOR]
```

- You can’t call features on entities using the **variant** mark that have a formal generic parameter type.

```
bus := school_bus
bus.extend (justin)
Invalid

vbus := school_bus
vbus.extend (justin)
Ok
```

```
bus := school_bus
bus.extend (justin)
Invalid
```

```
vbus := school_bus
vbus.extend (justin)
Ok
```