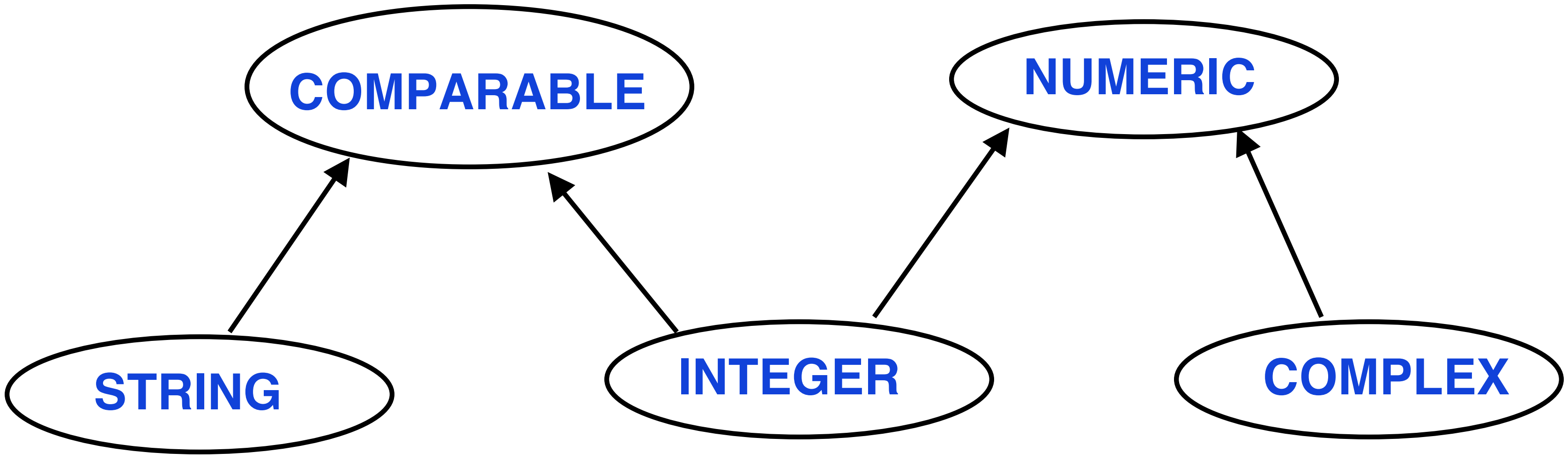
# Multiple & Repeated Inheritance

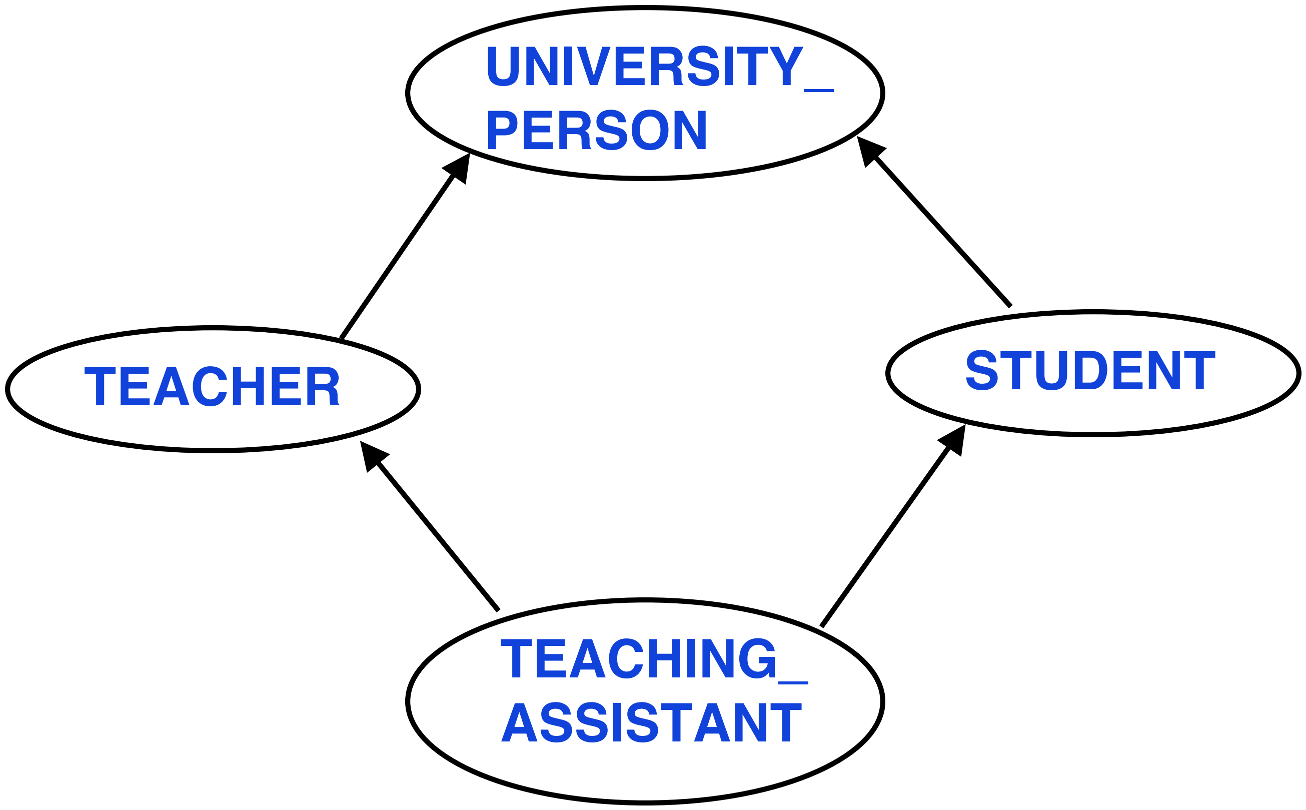
# Multiple Inheritance – Example

- Combining two abstractions into one
  - » **COMPARABLE and NUMERIC are both useful abstractions**
    - > **Some abstractions make use of both while others do not**



## Repeated Inheritance – Example

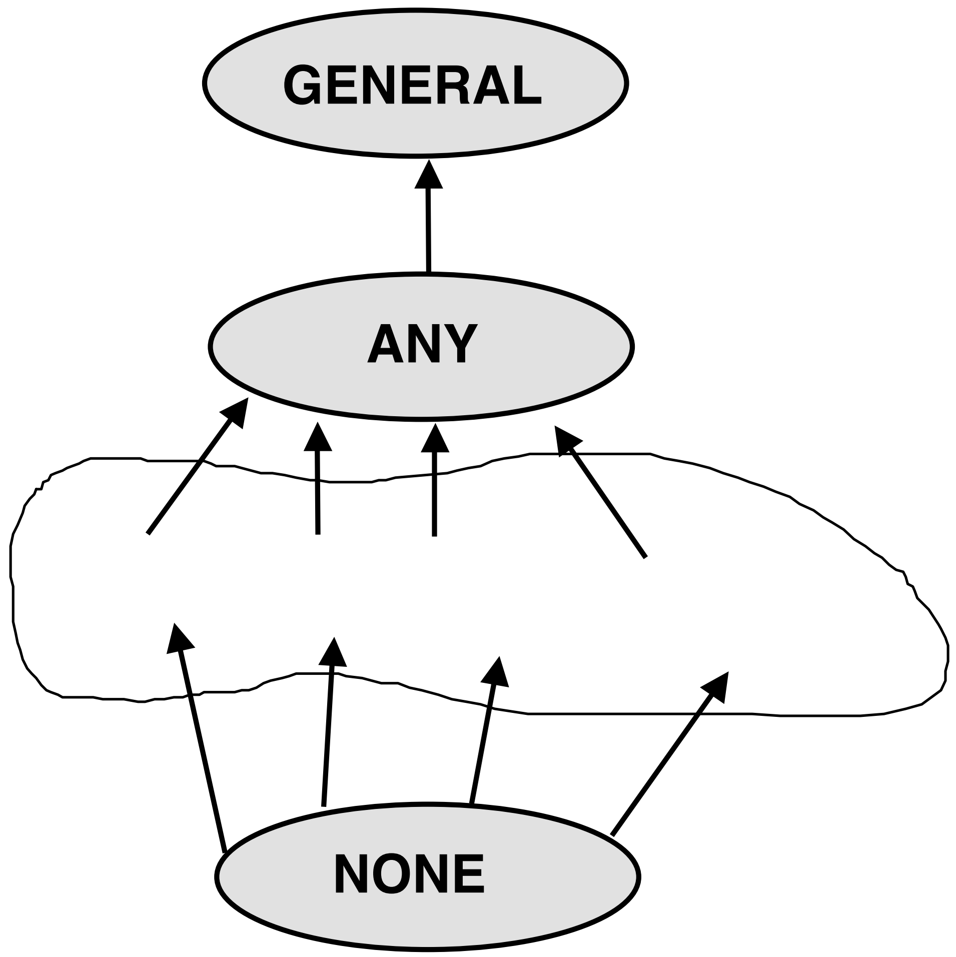
- Ancestor used in multiple paths to descendant



## Inheritance Types

- **Implementation** – abstraction that combines two implementations
  - » **ARRAY_STACK is both a STACK and and ARRAY**
- **Structural** – abstraction that combines two structures
  - » **HISTORY and STORABLE**

# Eiffel Global Inheritance Structure



GENERAL has all Eiffel global features & invariants

Customize ANY to have localized global features & invariants

## Feature Renaming

- Multiple & repeated inheritance lead to name clashes
  - **What if two parents use the same name for a feature?**
    - **A common occurrence since good names are reused**
  - **How can the child refer to the appropriate feature?**
- Answer
  - **Rename one of the features – give it an alias**
    - **Do not rely on overloading, not enough variation**
      - **Overloading - distinguishes features by argument type and count**

# Example Renaming

- Suppose **LONDON** and **LOS_ANGELES** both have the feature **foo**
- Then we can define **TORONTO** as follows

```
class TORONTO inherit
   LONDON  rename foo as fog
      redefine fog end
   LOS_ANGELES rename foo as smog
     redefine smog end
feature
    ...
end
```

# Renaming Effects

ldon : LONDON ; la : LOS_ANGELES ; tor : TORONTO

Valid – even after polymorphic assignment

```
ldon.foo  ;  tor.fog
la.foo    ;  tor.smog
```

Invalid

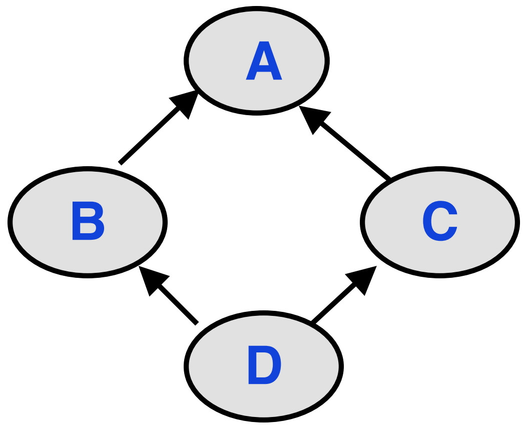
```
ldon.fog  ;  ldon.smog
la.fog    ;  la.smog
tor.foo
```

## Redeclaration & Renaming

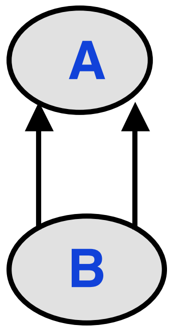
- Redeclaration
  - » **Keeps the name, changes the semantics**
- Renaming
  - » **Keeps the semantics changes the name**
- Can both rename and redefine
  - » **Rename first**
  - » **Use new name when redefining**
- Renaming can be useful to change the name to a more common one for the abstraction
  - » **TO push & pop (STACK) FROM add and remove (CONTAINER)**

## Repeated Inheritance

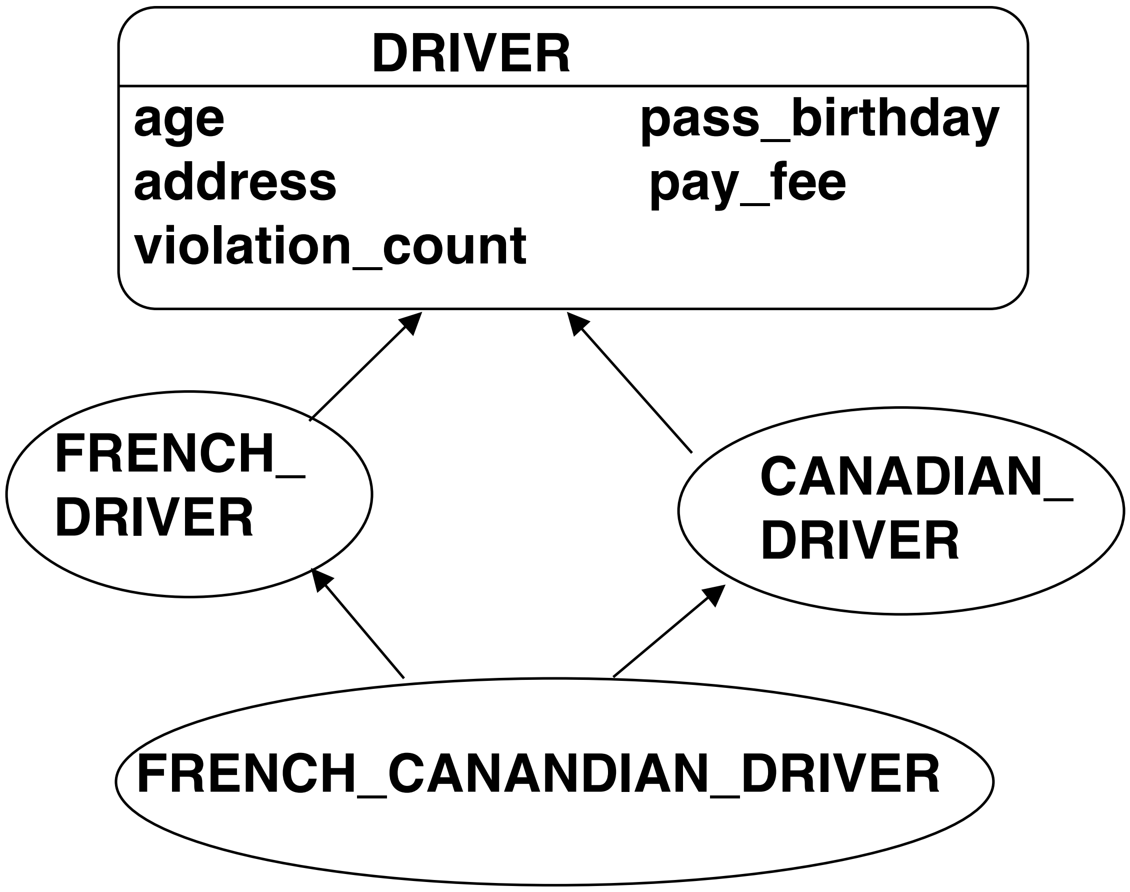
- Indirect
  - » class B inherit A
    class C inherit A
    class D inherit B C



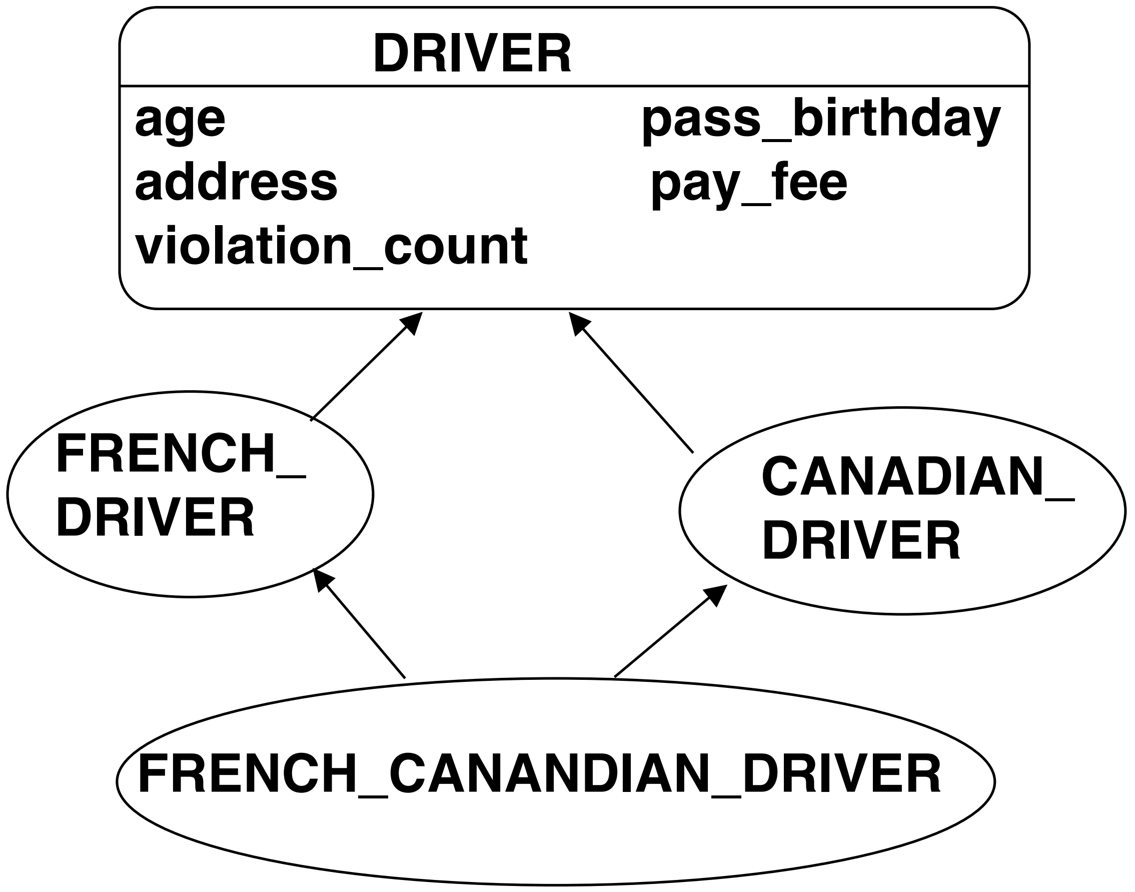
- Direct
  - » class B
      inherit
        A
        A



## Problems


DRIVER
age
address
violation_count
pass_birthday
pay_fee
FRENCH_
DRIVER
CANADIAN_
DRIVER
FRENCH_CANANDIAN_DRIVER

# Problems – 2



What about age?
It is the same for both drivers!

DO NOT rename!

Only rename if inheriting different but identically named features

Have a single shared feature

Sharing is not always appropriate
– violation_count, address, pay_fee –
are all different – need to replicate for each driver

## Repeated Inheritance Rule

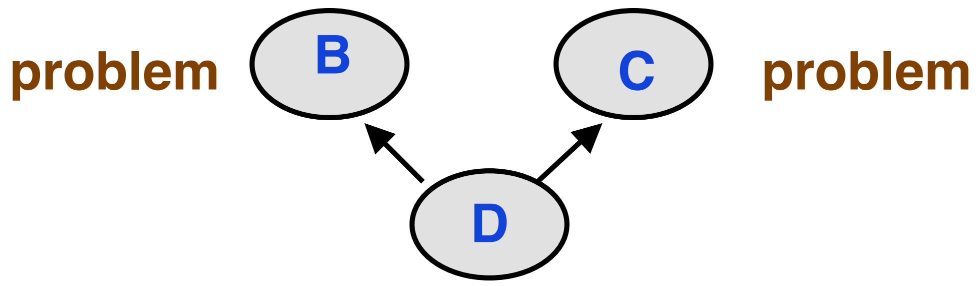
- In a repeated inheritance
  - » **Versions of a repeatedly inherited feature inherited under the same name represent a single feature**
  - » **Versions inherited under different names represent separate features, each replicated from the original in the common ancestor**
    - > **Use rename to get replication**
      - – **rename pay_fee as pay_french_fee**
- The rule applies to attributes as well as routines

## Single Name Rule

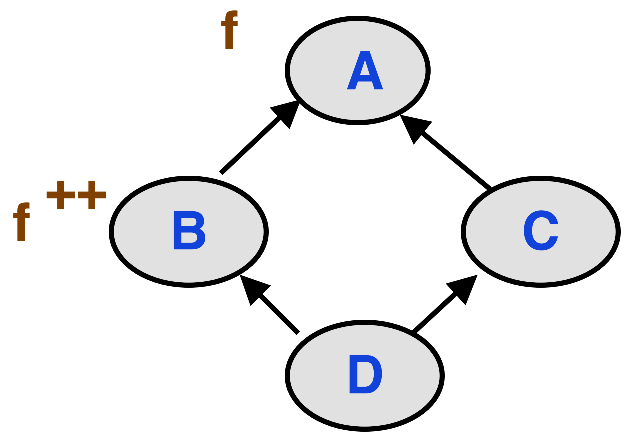
- Definition
  - » **The final name of a feature in a class is**
    - > **For an immediate feature, the name under which it is declared**
    - > **For an inherited feature that is not renamed, its final name is (recursively) in the parent from which it is inherited**
    - > **For a renamed feature, the name resulting from the renaming**
- Single Name Rule
  - » **Two different effective features of a class may not have the same final name**

## Must Rename

- Consider the following attributes, even if the types agree must rename **problem** in **D**
  - » **Rename either version from B or C or both**

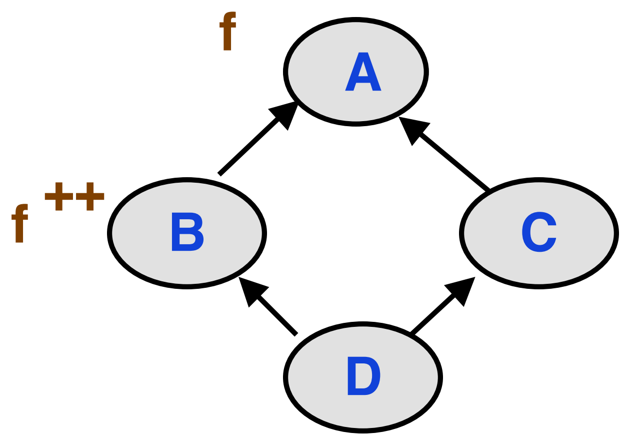


# Conflicting Redefinition



- In D have two different definitions of f
  - » From B and from A through C
- Consider under
  - » sharing
  - » replication

## Conflict Resolution – Sharing

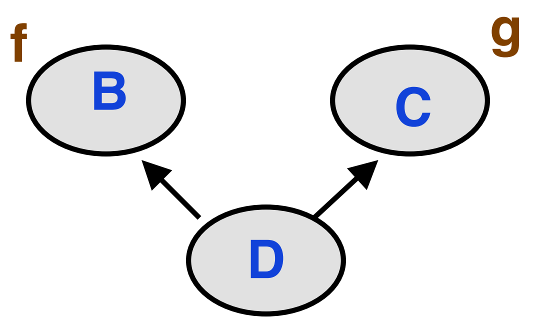


- Inherit under same name
  - » one version is deferred other is effective
    - > No problem – single name rule
  - » both versions effective but redefined in D
    - > No problem – produce one redefined version
  - » both effective, no redefinition
    - > Problem – name clash, must rename, get replication

# Conflict Resolution – Sharing – 2

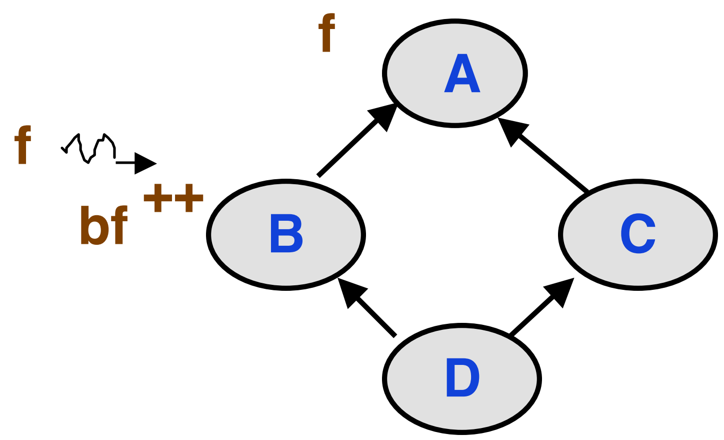
- Other solutions
  - » Make one of the versions deferred – Other takes over
    - > Could have intermediate class C' to defer
    - > Better is to use undefine
  - » Different names – join the solutions
    - > Requires compatible signatures and semantics

```
class D inherit
    B
    C undefine f end
....
```



```
class D inherit
    B
    C rename g as f
        undefine f end
....
```

# Conflict Resolution – Replication



- Suppose **a1** **:=** instance of **D**
  - » **Then a1.f is ambiguous**
    - > **could be either f or bf**
- Programmer must **select** the version

```
class D inherit
    B
    C select f end
....
```

```
class D inherit
    B select bf end
    C
....
```

## Select Rule

- A class that inherits two or more different effective versions of a feature from a repeated ancestor and does not redefine them both, must include exactly one of them in a **select clause**
  - » **Use select all if that is desired**

## Genericity with Repeated Inheritance

- The type of any feature that is **shared** under the repeated inheritance rule, and the type of any of its arguments if it is a routine, may not be a generic parameter of the class from which the feature is repeatedly inherited

```
class A[G]  feature                class B inherit
   f : G                              A [INTEGER]
end                                   A [REAL]
                                   end
```

  - » **Ambiguity as to the type for f in B.**
  - » **Use renaming to get replication, if genericity is needed**

## Name Clashes – Definition & Rule

- In a class obtained through multiple inheritance, a **name clash** occurs when two features inherited from different parents have the same final name
- A **name clash** makes the class **invalid** **except** in any of the following cases
  - » **The two features are inherited from a common ancestor and none has been redeclared from the version in that ancestor**
  - » **Both features have compatible signatures and at least one of them is inherited in deferred form**
  - » **Both features have compatible signatures and they are both redefined in the class**
    - > **As one redefinition for the feature**

## Summary of Adaptation Clauses

- Eiffel adaptation  clauses are in the following order.

```
class B
  inherit A
    rename f1 as new_f1, f2 as new_f2, f3 as new_f3
    export {A, B} new_f1, f4
    undefine new_f3, f6
    redefine new_f2, f5
    select new_f2, f7
  end
```