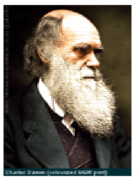
Darwin & Lava

– Object-based Dynamic Inheritance ... in Java –



Günter Kniesel
University of Bonn, Computer Science Department III
Römerstraße 164, D-53117 Bonn
Contact: Guenter.Kniesel@cs.uni-bonn.de

JavaLab

Why dynamic inheritance?

Rigidity Problem
- Class-based inheritance is too rigid.
- It cannot express dynamically evolving object structure and behaviour.
- The inherited interface and the inherited code are determined at compile-time.

Solution
- Only a part of the inherited interface is declared statically.
- The inherited code and part of the inherited interface is determined at run-time.

Why object-based inheritance?

Granularity Problem
- Class-based inheritance is too coarse-grained.
- It cannot express *variations of structure and behaviour* among instances of one class.
- It cannot express *sharing of state* between objects.

Solution
- Differences between instances of the same class can be expressed by letting them *inherit behaviour and state from instances* of other different classes.

Delegation

- What is "Delegation"?
  - "Delegation" is a shorthand for "object based, dynamic inheritance".
- General Syntax
  - object variable annotation **`delegatee`**
- General Effects
  - automatic forwarding of method and variable accesses to delegatee
  - binding of "**`this`**" to delegator within forwarded method invocations
  - overriding of delegatee's methods by delegator
  - the delegator class is a subtype of the declared delegatee type (e.g. in all examples, EuroWrapper is a subtype of Finance)

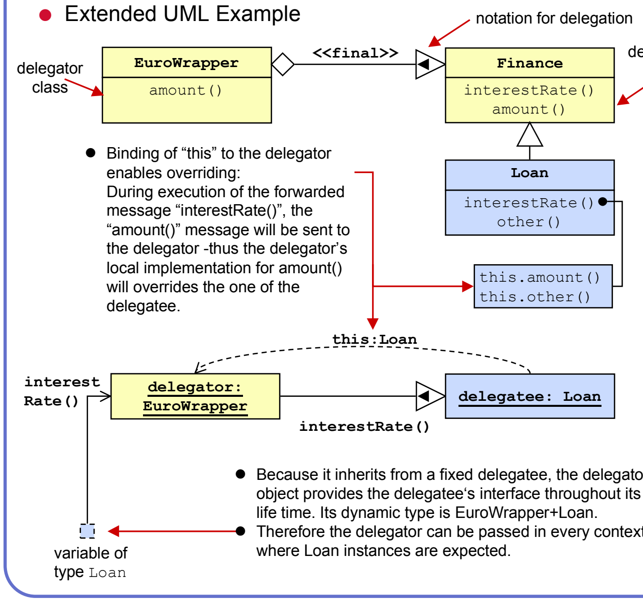
Fixed Delegation

- Syntax
  - object variable annotation **`final delegatee`**
- Additional Effects
  - after instantiation the delegator never changes its delegatee
  - the delegator object's type is a subtype of the delegatee object's type
- Code Example

```
// EuroWrapper adapts Finance objects to Euro
// calculation by redefining their amount() method
class EuroWrapper {
    // Everything that is not declared locally will
    // be delegated automatically to _delegatee:
  ... final delegatee Finance _delegatee;
    // overriding behaviour:
  public double amount(){
     _delegatee<-amount() / ExchangeRate;
   }
}
```

explicit delegation (equivalent of `super` call)

- Extended UML Example

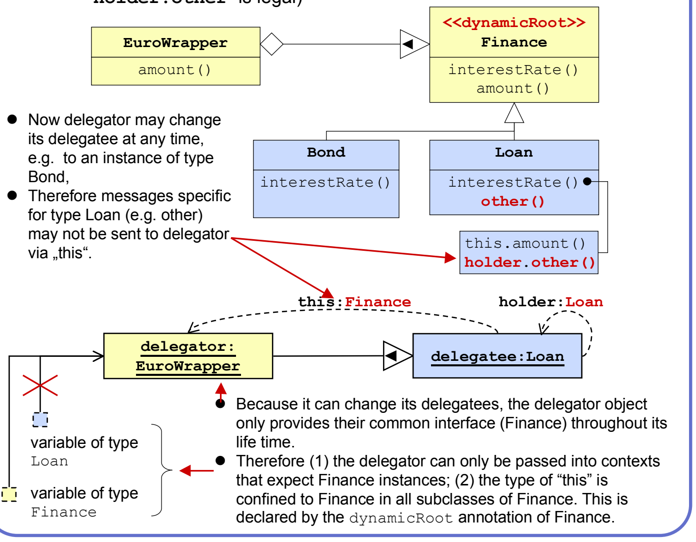


- Binding of "this" to the delegator enables overriding:
During execution of the forwarded message "interestRate()", the "amount()" message will be sent to the delegator -thus the delegator's local implementation for amount() will overrides the one of the delegatee.

- Because it inherits from a fixed delegatee, the delegator object provides the delegatee's interface throughout its life time. Its dynamic type is EuroWrapper+Loan.
- Therefore the delegator can be passed in every context where Loan instances are expected.

Mutable Delegation

- Additional Problem
  - Mutable delegation allows delegators to change their delegatees at any time.
  - Therefore messages specific for one particular delegatee type (e.g. Loan) may not be sent to delegators via "**`this`**".
- Solution: "Split self" approach
  - "**`this`**" refers to the object that initially received the delegated message
  - "**`holder`**" refers to the object that executes the current method
  - annotation "**`dynamicRoot`**" for classes and interfaces
    - "**`this`**" has the annotated type in all subclasses (e.g. in the example below it has type Finance in class Finance *and all its subclasses*)
    - therefore methods that are not in the annotated type cannot be invoked on **`this`** (e.g. **`this.other`** would be illegal in class Loan),
    - specific local methods can be invoked on **`holder`**, which is always of local type (e.g. in class Loan holder has type Loan, so **`holder.other`** is legal)



- Now delegator may change its delegatee at any time, e.g. to an instance of type Bond,
- Therefore messages specific for type Loan (e.g. other) may not be sent to delegator via „this".

- Because it can change its delegatees, the delegator object only provides their common interface (Finance) throughout its life time.
- Therefore (1) the delegator can only be passed into contexts that expect Finance instances; (2) the type of "this" is confined to Finance in all subclasses of Finance. This is declared by the `dynamicRoot` annotation of Finance.

Darwin and Lava

- What is Darwin?
  - a model for class-based, statically typed, object-oriented languages with object-based, dynamic inheritance (delegation)
- What is Lava?
  - an extension of Java with object-based dynamic inheritance (delegation)
- Contributions
  - static type-safety of object-based, dynamic inheritance
  - easy modelling of object-specific behaviour and behaviour evolution
    - non-monotonic (new behaviour can be *acquired and abandoned*)
    - unanticipated, identity-changing (wrapper-/decorator-style)
    - anticipated, identity-preserving evolution (state-/strategy-style)
- More Information
  - **http://javalab.cs.uni-bonn.de/research/darwin/**