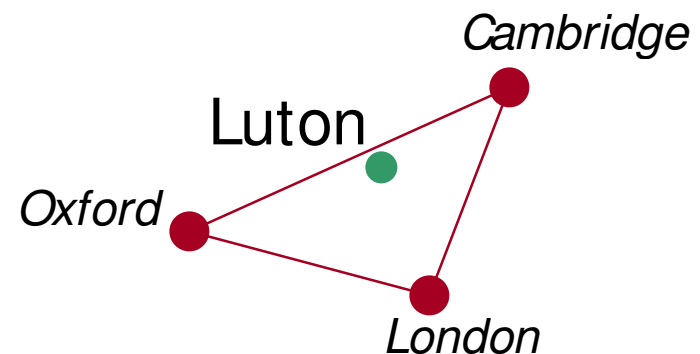


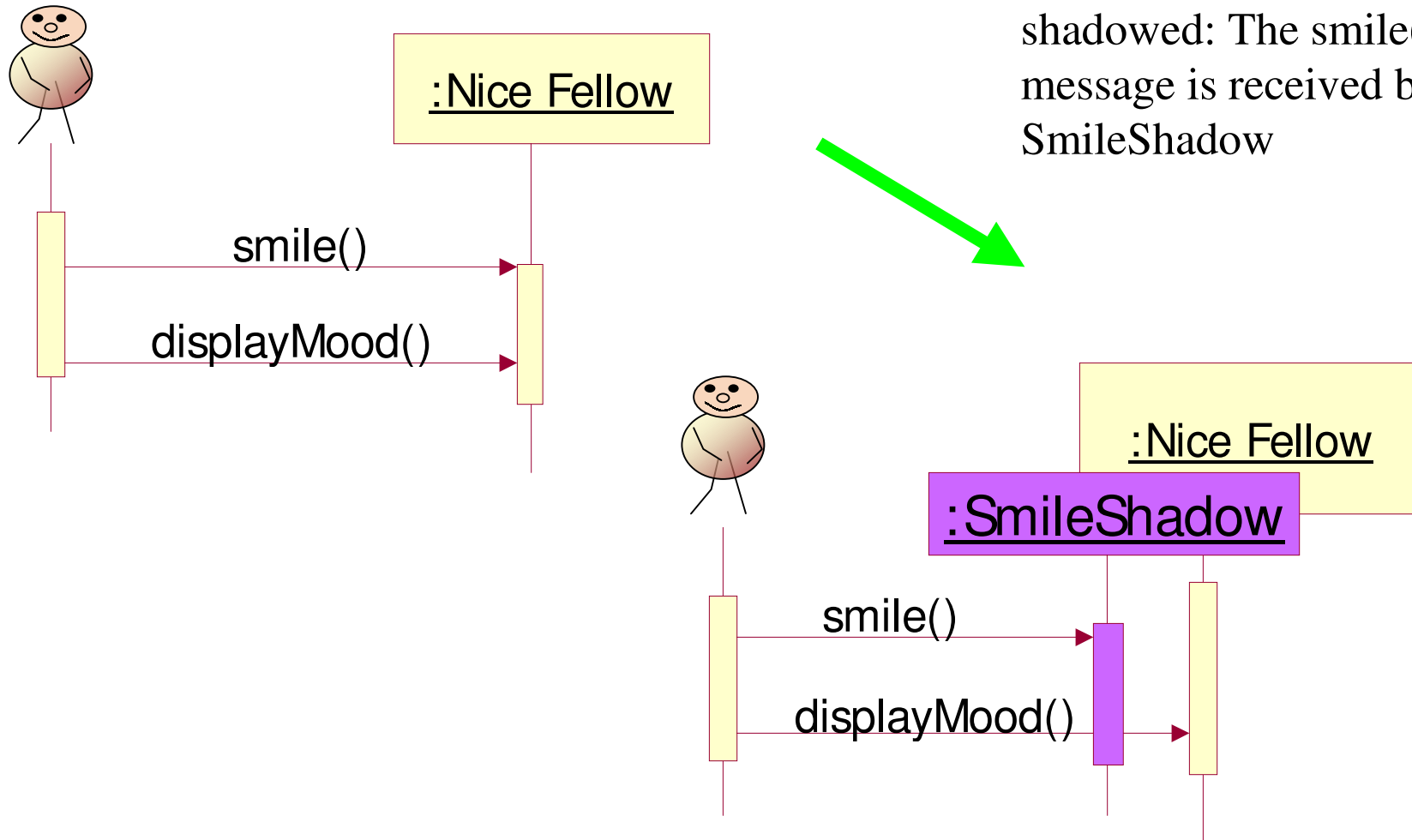
Object Shadowing – a Key Concept for a Modern Programming Language

- by:
 - Marc Conrad
 - Tim French
 - Carsten Maple
- University of Luton, UK.



Simple Example for a Shadow

- The Nice Fellow object is shadowed: The smile() message is received by the SmileShadow





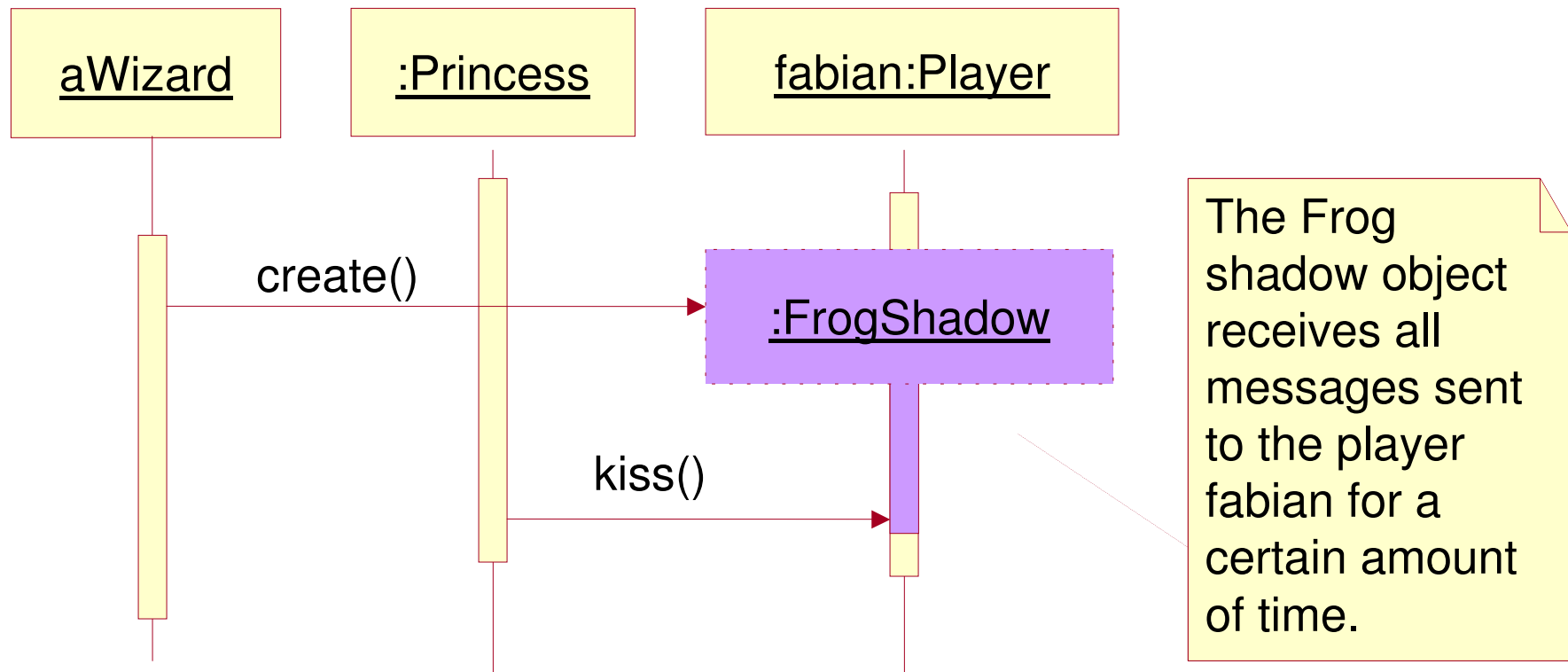
What are Shadows?

- Shadows are known in LPC, an OO language for MUDs (text based computer games).
- They change the **behaviour** of an object for a certain amount of time (E.g. “magic spell” of a wizard).
- Usage is highly **pragmatic**.
- So far **no academic evaluation** of the concept.
- Experimental implementation in Java by the authors at <http://perisic.com/shadow>

- Note: The feature is known in LPC as “shadow”, however a different name may be more appropriate, e.g. “dynamic decorator”, or “generalised traits”, etc.

Typical example from Computer Games

- A *wizard* makes a spell and changes *fabian* into a frog. An instance of a *princess* kisses *fabian* and changes him back.





Some Properties (*to be discussed*)

- Shadows cannot be shadowed but an object can have multiple shadows.
- An object can decide not to allow shadows, or to disallow certain methods from being shadowed.
- Attributes cannot be shadowed.
- Only calls from external objects can be shadowed (=> different behaviour of *bla()* and *this.bla()*).

- *Note: This is how shadows are implemented in LPC. However a different behaviour may be more appropriate in other contexts.*
- *A formal, precise definition of the semantics would be desirable as well.*

Application I

Deprecating Methods



Deprecated pipe

- Problem:
 - Software is under constant evolution.
 - How to remove deprecated methods?
 - E.g. Java: all deprecated methods are still there for backward compatibility.
- Solution:
 - Provide deprecated methods in shadows and keep the library free from them.
 - E.g. Specify required shadows for a class in environment variables, then the class loader decides when to add a shadow.



Application II

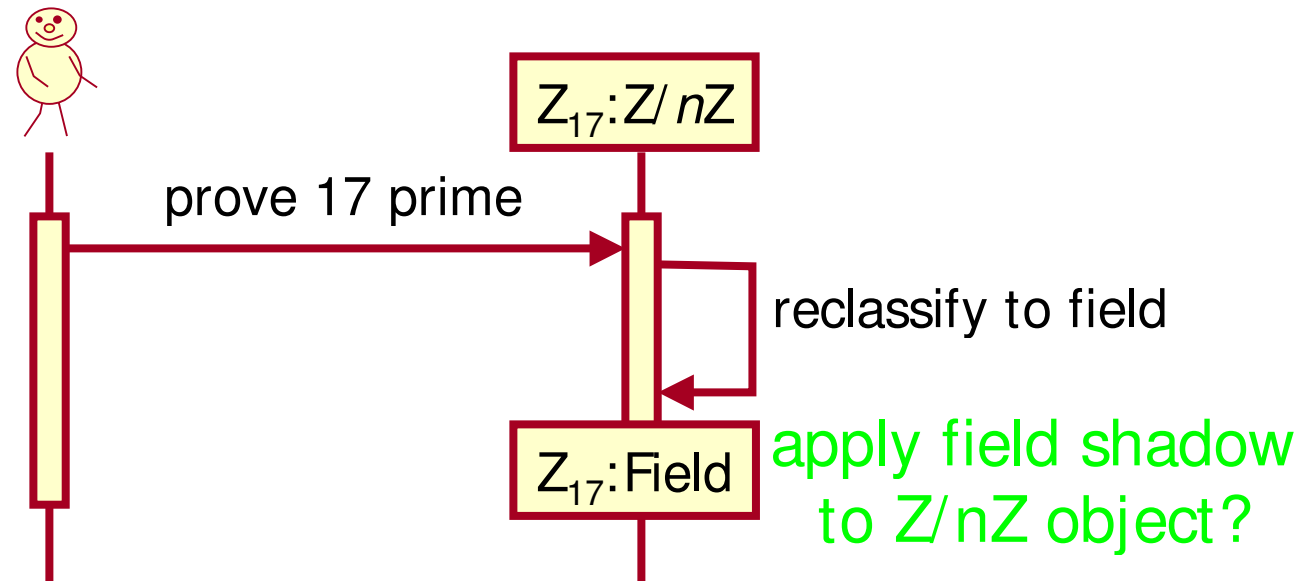
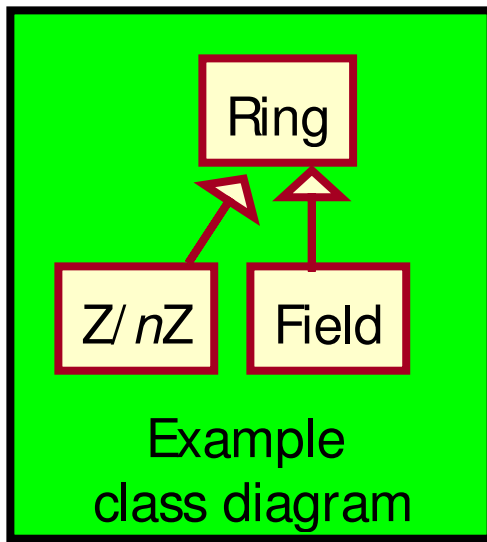
Prototyping

- Problem:
 - An existing library is to be further developed, but should not / cannot be changed.
 - E.g. copyright issues, change only valid for a very specific application area, stable versions vs. test versions, etc.
- Solution:
 - A shadow adds additional behaviour **temporarily** to an existing class.
 - “Inverse Deprecation”.

Application III

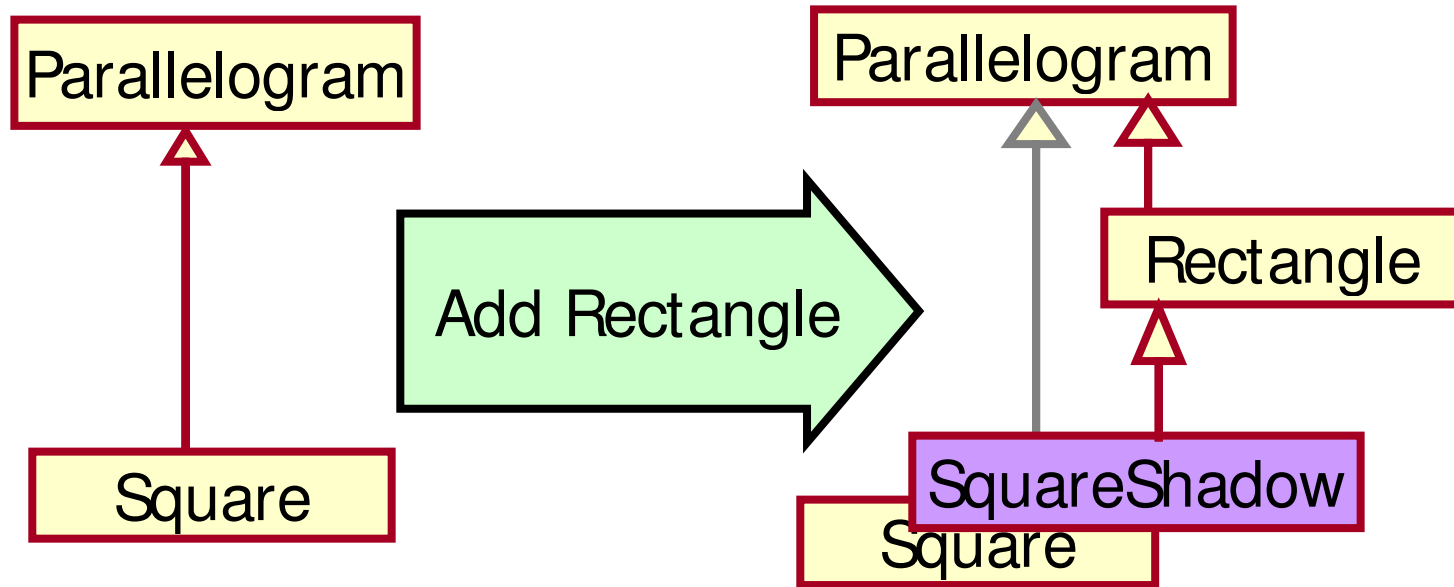
Reclassification

- Problem:
 - An object of class A adopts the behaviour of a class B during run-time.
- Solution:
 - Shadows change behaviour at run-time!



Application IV

Interclassing



- Problem:
 - A class Rectangle is inserted in the middle of a Parallelogram/Square inheritance relationship.
- Solution:
 - Shadow the child class.

Application V

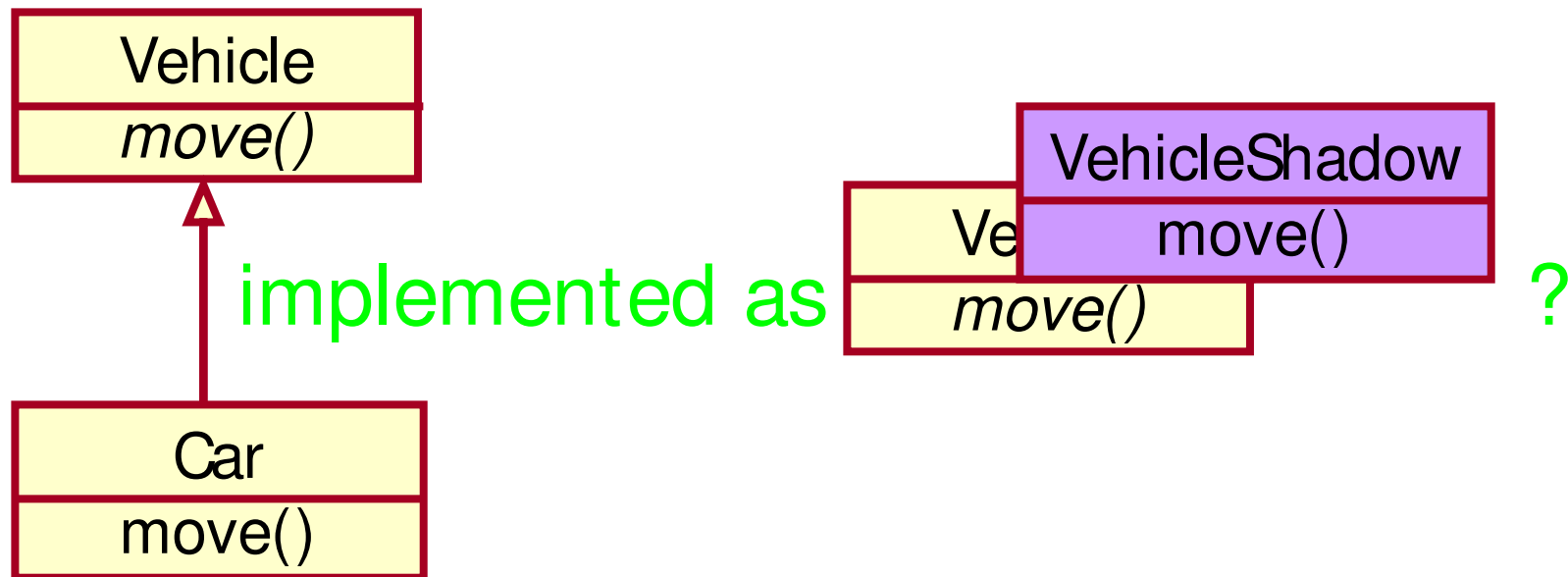
Shadows and Inheritance (provocative)

- Problem:

- There is no problem here, but let's have a look anyway.

- Solution:

- Implement inheritance itself via shadows.





Summary

- Shadows exist and are useful in the application area of MUDs
- However they are not discussed in a wider OO context.
- Being available as an easy to use standard feature in a mainstream programming language shadows can be applied for **deprecating methods**, **prototyping**, **reclassification**, **interclassing**, and the **implementation of inheritance itself**.
- If shadows are so useful, why are they not available in mainstream languages?