Ca' Foscari
University
of Venice

Value and reference types

Object oriented programming, module 1

Pietro Ferrara
pietro.ferrara@unive.it

Summary of previous lectures

- Classes are composed by data (fields) and functionalities (methods)
- Classes are instantiated into objects
- Constructors initialize objects
- Fields and methods can be modified by
  - static: referring to the whole class and not to a single instance
    - Shared among all the instances of the class
  - final (only fields): assigned during initialization, then cannot be reassigned
- Methods provide functionalities of a class
  - Need to operate on the status of the current (this) object

What a class is

- A class defines the structure of objects
- Class Car contains
  - Fields speed, fuelType, fuel
    - of type double, FuelType, and double, respectively
  - Methods refuel, accelerate, fullBreak
    - not returning any value
    - receiving a FuelTank object, a double value, and nothing as parameters, respectively
- E.g., like a table structure in Excel/DB

```
class Car {
  double speed;
  FuelType fuelType;
  double fuel;
  void refuel(FuelTank tank) {…}
  void accelerate(double a) {…}
  void fullBreak() {…}
}
```

What an object is

- **An object is an instance of a class**
- An object contains concrete values for each field
- A class can be instantiated many times
  – Each instance has its own state
  – That is, different values for each field
  – Remember: static fields are different!

```
class Car {
  double speed;
  FuelType fuelType;
  double fuel;
  void refuel(FuelTank tank) {…}
  void accelerate(double a) {…}
  void fullBreak() {…}
}

Car c = new Car(100, “diesel“, 10);
c.accelerate(10);
```

new operator

- The new operator is followed by a class name and it instantiates the class
  1. Allocates the memory to store the state of an object
  2. Initializes all fields to zeros/null
  3. Invoke the specified constructors
  4. It returns a pointer to an object
- Note that for each field a memory location is allocated in the heap
  – Fields occupies memory permanently!

```
class Car {
  double speed;
  FuelType fuelType;
  double fuel;
  void refuel(FuelTank tank) {…}
  void accelerate(double a) {…}
  void fullBreak() {…}
}


Car c = new Car(100, “diesel“, 10);
c.accelerate(10);
```

Local variables, parameters, and fields

Fields

- Defined in the class body
- Accessible by all class methods and constructors
- Persist during the object lifecycle

Parameters

- Defined in the method signature
- Accessible by the method body
  - Can be assigned, but it's a bad practice
- Value lost after the method execution

Local variables

- Defined in the method body
- Accessible by the method body
- Value lost after the method execution

```
class FuelTank {
 FuelType type;
 double amount;
 double cost;
 FuelTank(
   FuelType type,
   double amount,
   double costPerLiter) {
   this.type = type;
   this.amount = amount;
   double overallCost =
    amount * costPerLiter
   this.cost = overallCost;
 }
}
```

Reference and value types

- We can split Java types into 2 main categories
  - Value (aka, primitive) types: int/long, float/double, char, booleans
  - Reference types: objects, arrays, …
- What's the difference? Like C:
  - Reference types contain a pointer
    - Side effects when modifying the state of the object
  - Value types contain the value
  - C asterisks are hidden under the hood

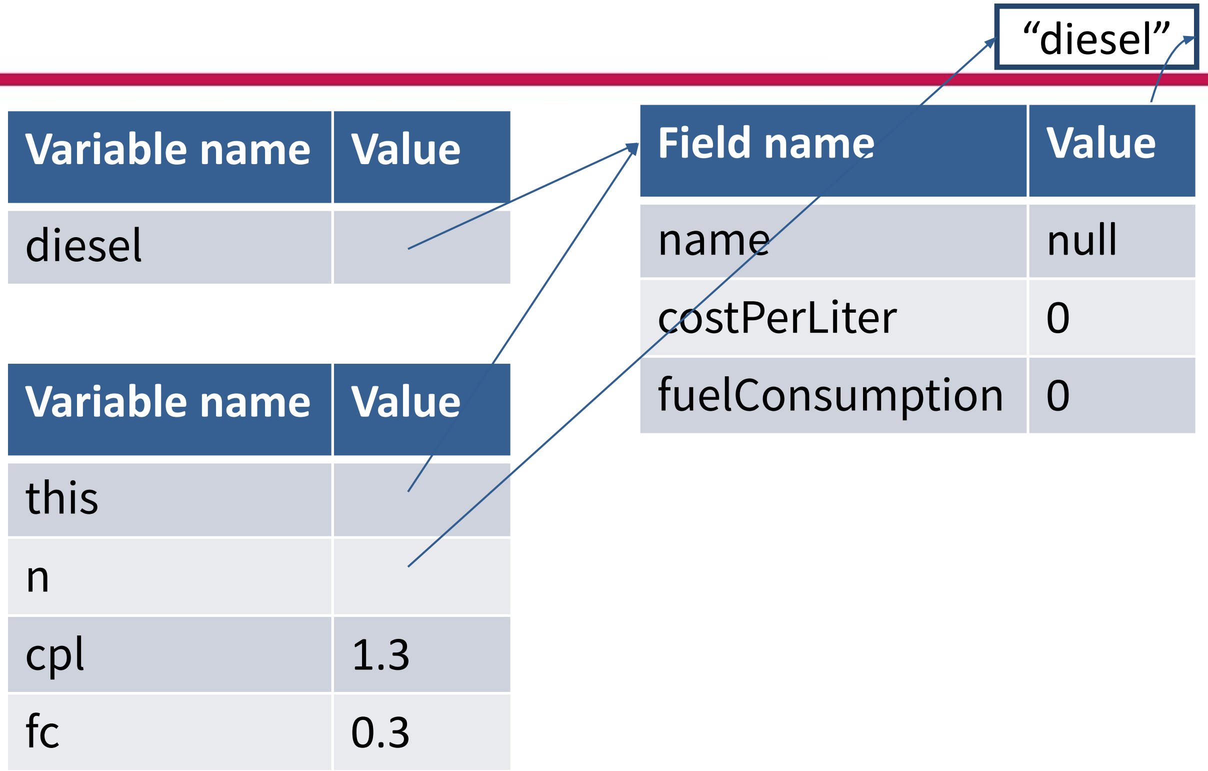
```
class Car {
  double speed;
  FuelType fuelType;
  double fuel;
  void refuel(FuelTank tank) {
    fuel += tank.amount;
    tank.amount = 0;
  }
}


c.refuel(tank)
//tank.amount == 0!
```

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```



Variable name	Value
diesel	

Variable name	Value
this	
n	
cpl	1.3
fc	0.3

Field name	Value
name	null
costPerLiter	0
fuelConsumption	0

“diesel”

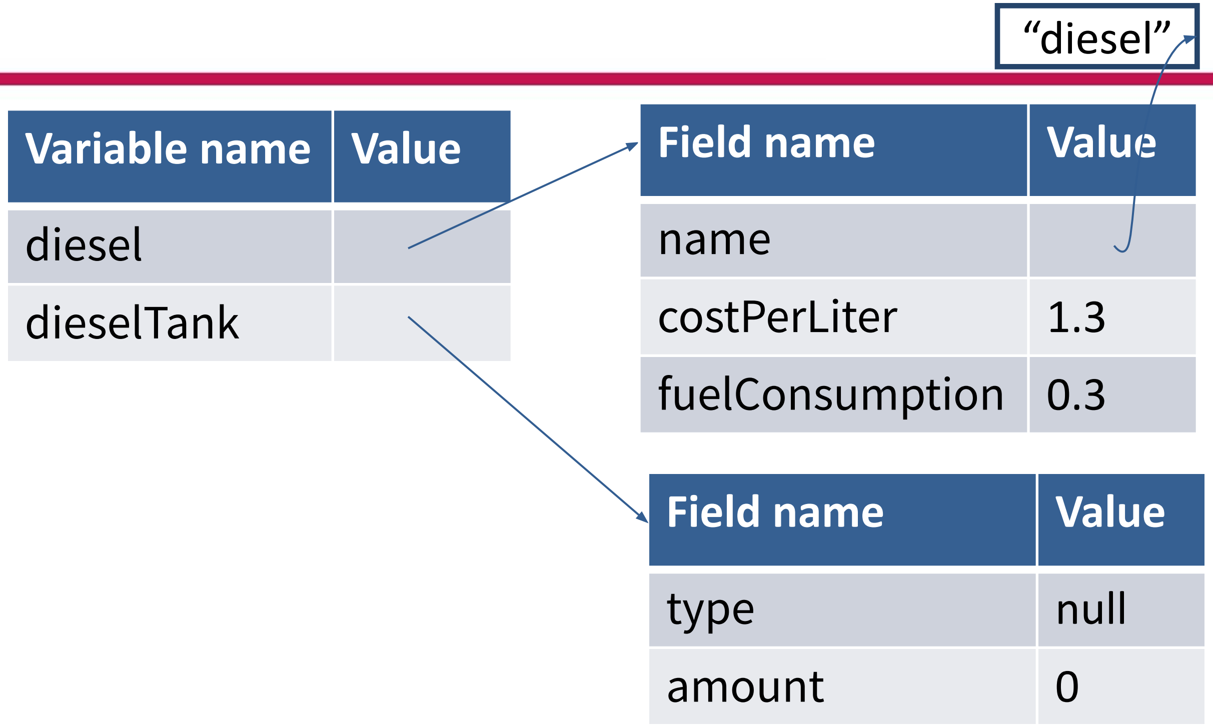
```
class FuelType {
  String name;
  double costPerLiter;
  double fuelConsumption;
}
```

```
FuelType(String n, double cpl, double fc) {
  name = n;
  costPerLiter = cpl;
  fuelConsumption = fc;
}
```

Ca' Foscari University of Venice

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```



Variable name	Value
diesel	
dieselTank	

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

“diesel”

Field name	Value
type	null
amount	0

```
class FuelType {
  String name;
  double costPerLiter;
  double fuelConsumption;
}
```

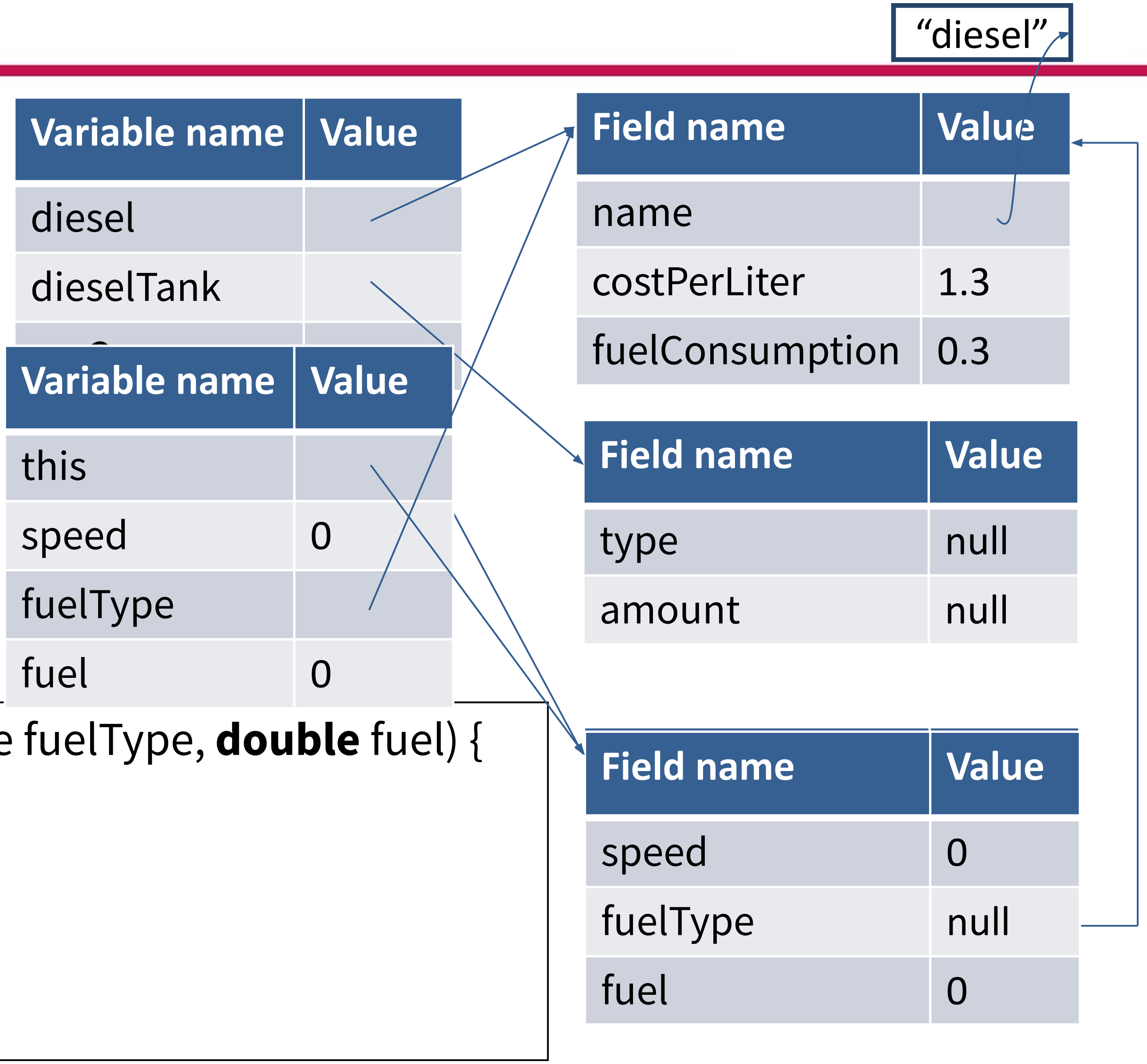
```
class FuelTank {
  FuelType type;
  double amount;
}
```

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```

```
class Car {
 double speed;
 FuelType fuelType;
 double fuel;
}
```

```
Car(double speed,  FuelType fuelType, double fuel) {
  this.speed = speed;
  this.fuelType = fuelType;
  this.fuel = fuel;
}
```



Variable name	Value
diesel	
dieselTank	

Variable name	Value
this	
speed	0
fuelType	
fuel	0

“diesel”

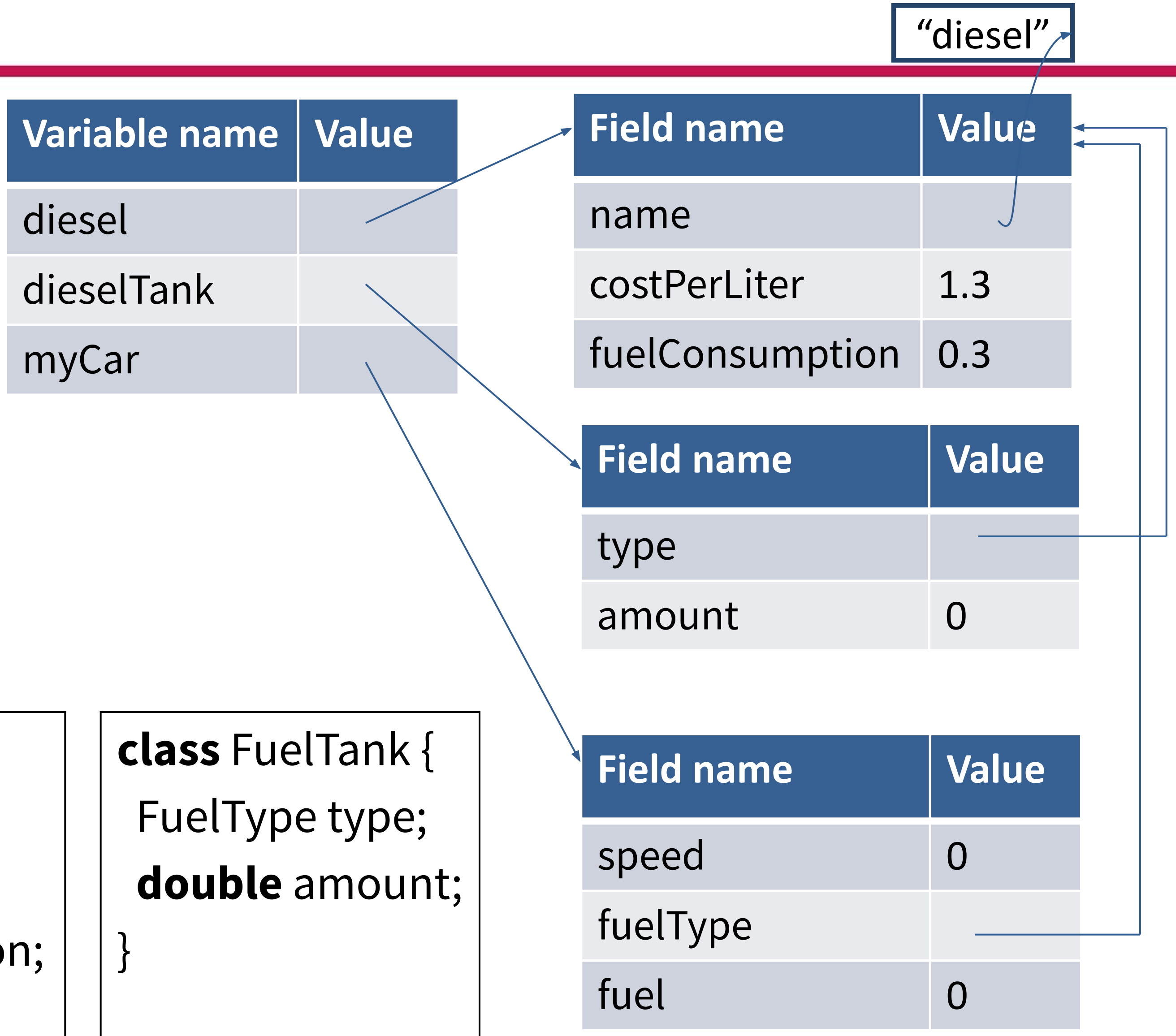
Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	null
amount	null

Field name	Value
speed	0
fuelType	null
fuel	0

A complete example

```
FuelType diesel = new FuelType("diesel", 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```



“diesel”

Variable name	Value
diesel	
dieselTank	
myCar	

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	0

Field name	Value
speed	0
fuelType	
fuel	0

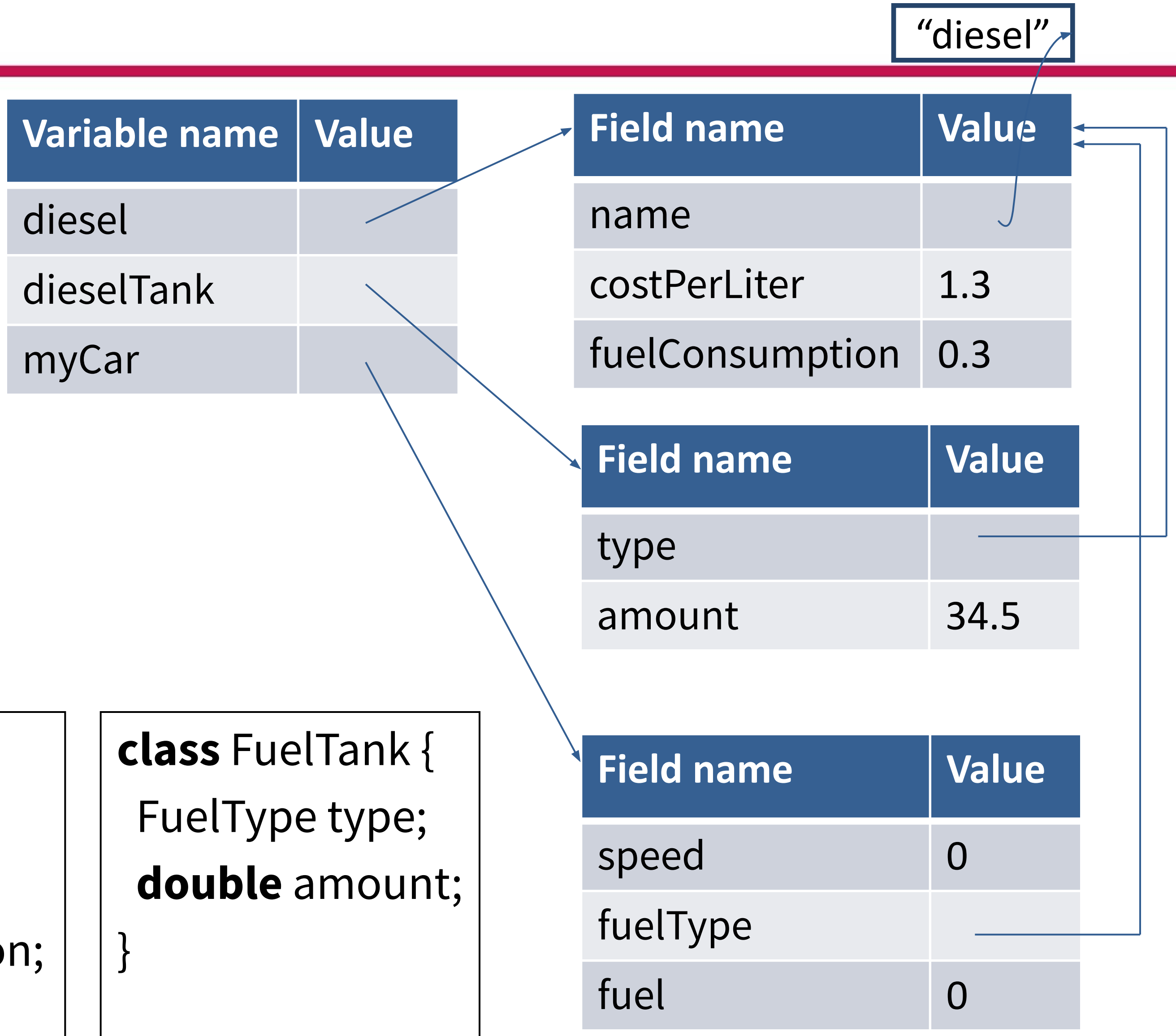
```
class Car {
 double speed;
 FuelType fuelType;
 double fuel;
}
```

```
class FuelType {
 String name;
 double costPerLiter;
 double fuelConsumption;
}
```

```
class FuelTank {
 FuelType type;
 double amount;
}
```

A complete example

```
FuelType diesel = new FuelType("diesel", 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```



"diesel"

Variable name	Value
diesel	
dieselTank	
myCar	

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	34.5

Field name	Value
speed	0
fuelType	
fuel	0

```
class Car {
 double speed;
 FuelType fuelType;
 double fuel;
}
```

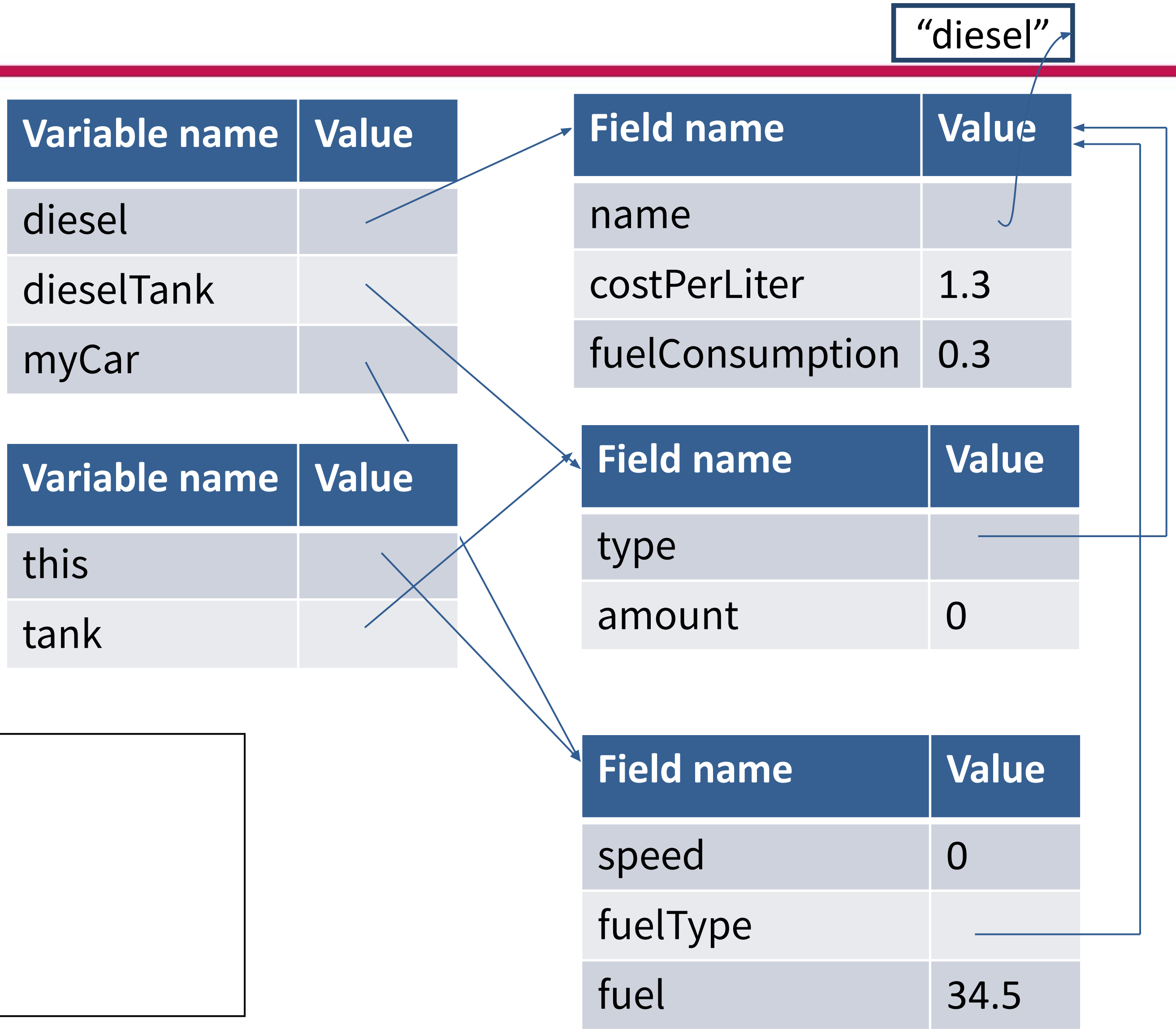
```
class FuelType {
 String name;
 double costPerLiter;
 double fuelConsumption;
}
```

```
class FuelTank {
 FuelType type;
 double amount;
}
```

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```

```
void refuel(FuelTank tank) {
  fuel += tank.amount;
  tank.amount = 0;
}
```



“diesel”

Variable name	Value
diesel	
dieselTank	
myCar	

Variable name	Value
this	
tank	

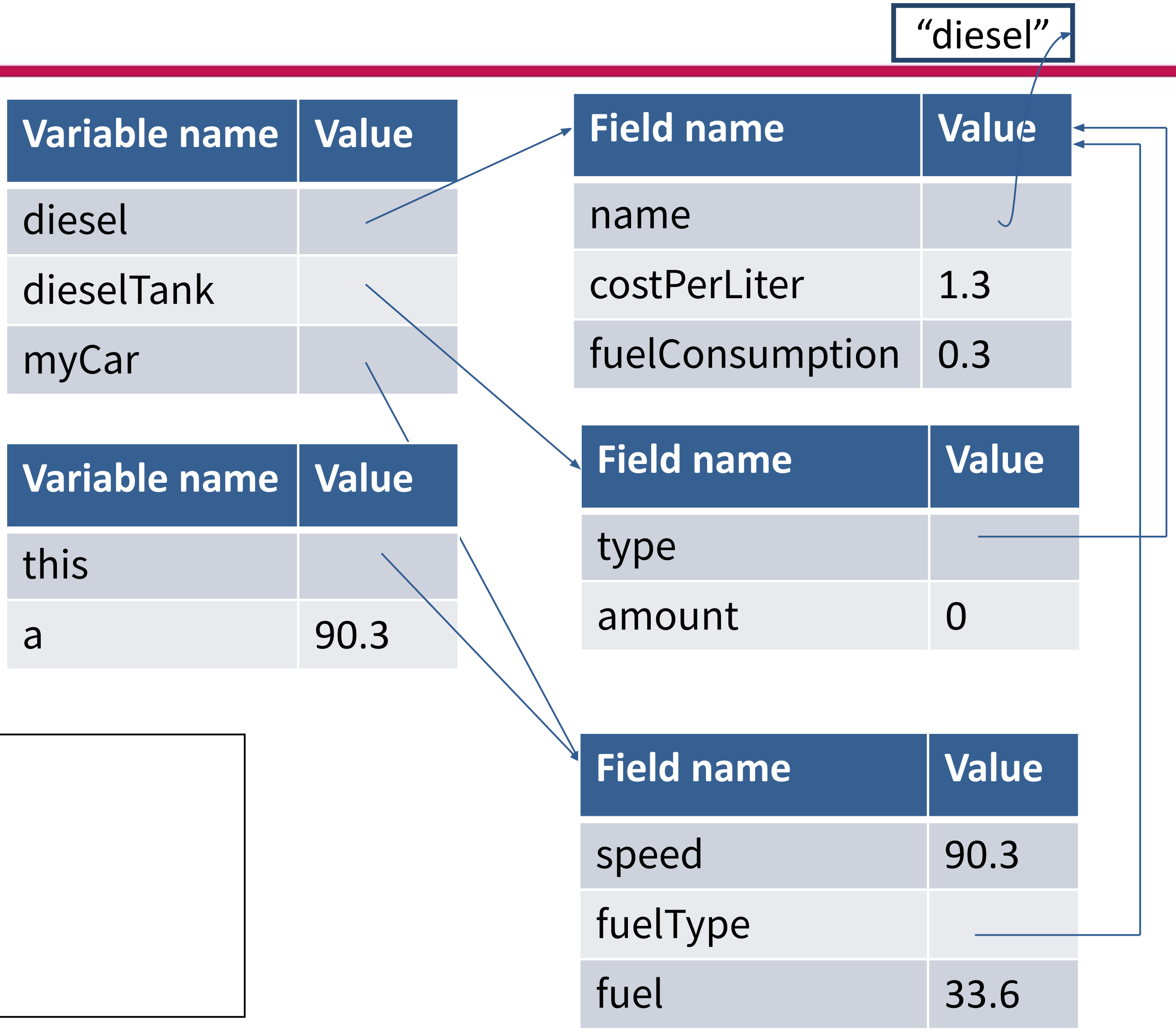
Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	0

Field name	Value
speed	0
fuelType	
fuel	34.5

A complete example

```
FuelType diesel = new FuelType(“diesel“, 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```



Variable name	Value
diesel	
dieselTank	
myCar	

Variable name	Value
this	
a	90.3

“diesel”

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	0

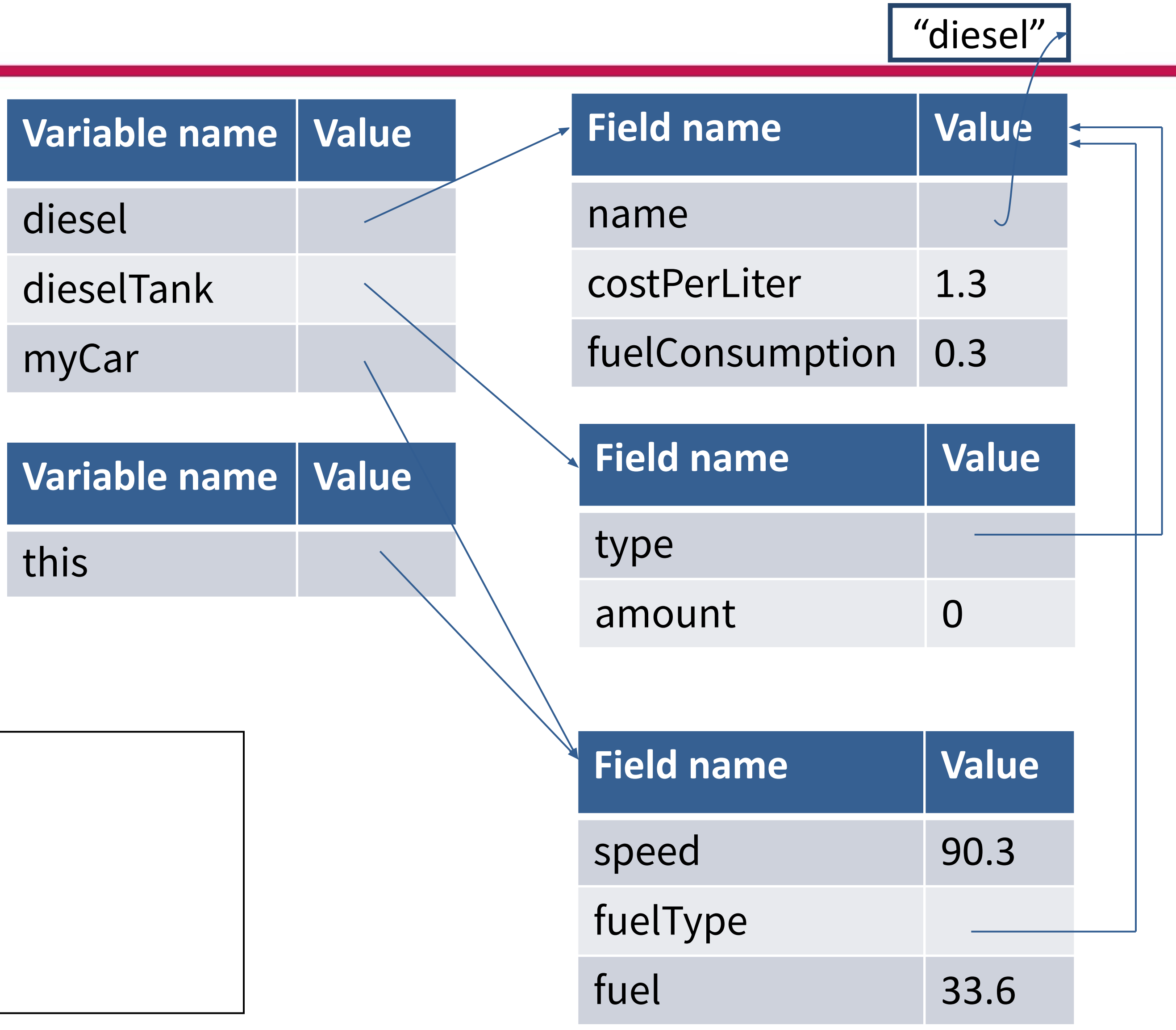
Field name	Value
speed	90.3
fuelType	
fuel	33.6

```
void accelerate(double a) {
  speed += a;
  fuel -= a*0.01;
}
```

A complete example

```
FuelType diesel = new FuelType("diesel", 1.3, 0.3);
FuelTank dieselTank = new FuelTank();
Car myCar = new Car(0, diesel, 0);
dieselTank.type = diesel;
dieselTank.amount = 34.5;
myCar.refuel(dieselTank);
myCar.accelerate(90.3);
myCar.fullBreak();
```

```
void fullBreak() {
  speed = 0;
}
```



"diesel"

Variable name	Value
diesel	
dieselTank	
myCar	

Variable name	Value
this	

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	0

Field name	Value
speed	90.3
fuelType	
fuel	33.6

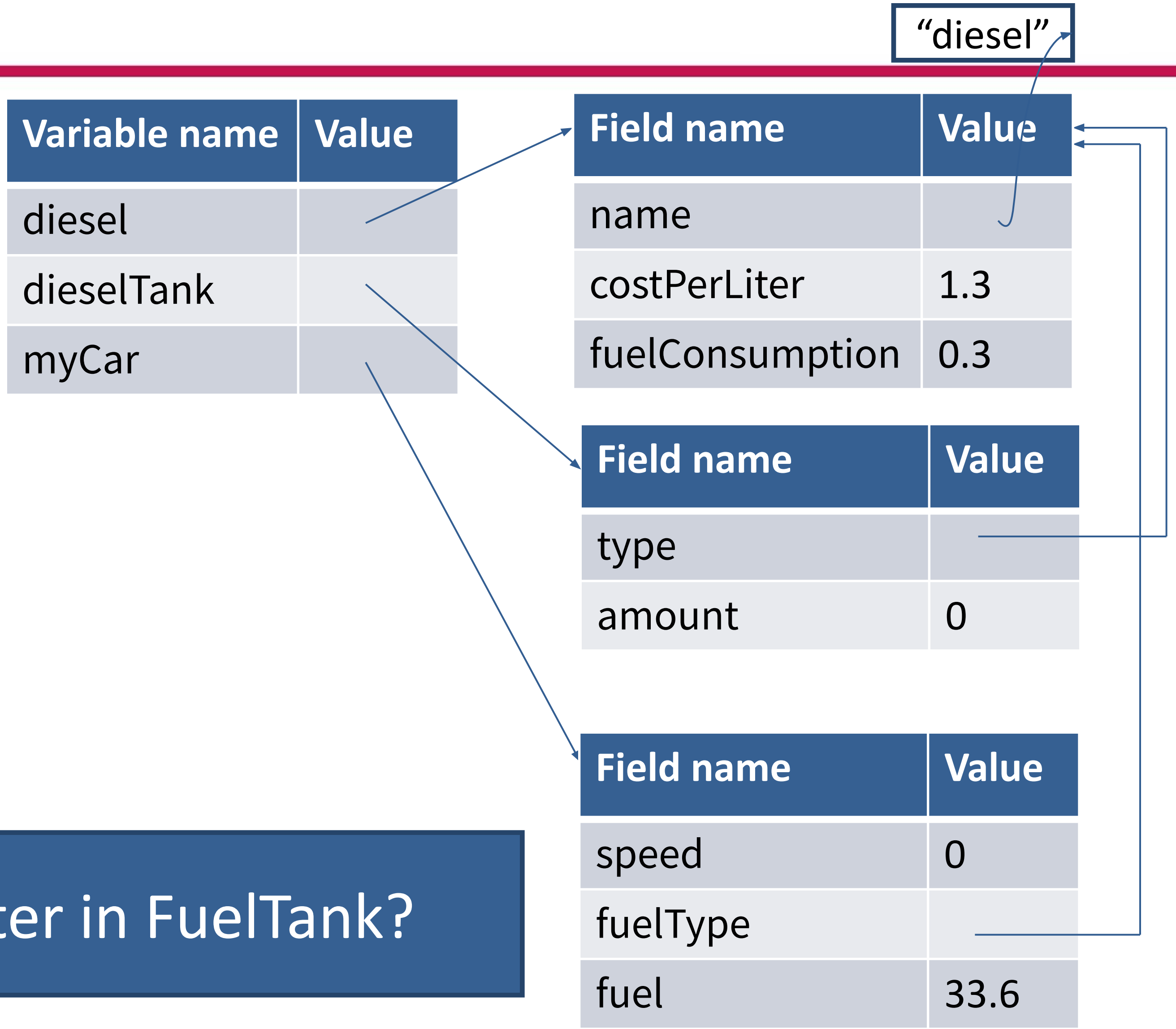
A complete example

Change the price of diesel to 1.35

```
diesel.costPerLiter = 1.35
```

```
class FuelType {
  String name;
  double costPerLiter;
  double fuelConsumption;
}
```

```
class FuelTank {
  FuelType type;
  double amount;
}
```



"diesel"

Variable name	Value
diesel	
dieselTank	
myCar	

Field name	Value
name	
costPerLiter	1.3
fuelConsumption	0.3

Field name	Value
type	
amount	0

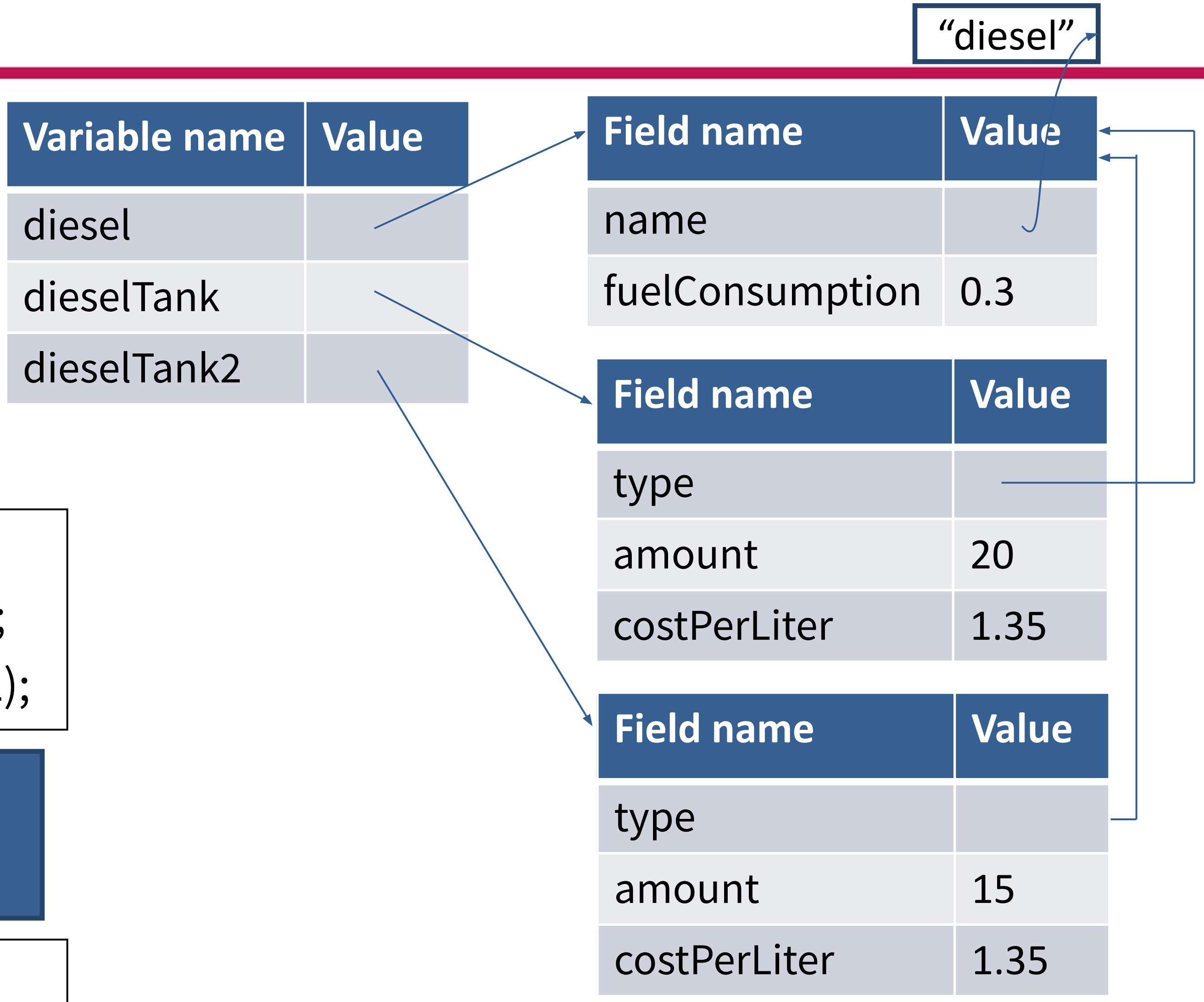
Field name	Value
speed	0
fuelType	
fuel	33.6

What happens if we have costPerLiter in FuelTank?

A complete example

```
class FuelType {
  String name;
  double fuelConsumption;
}
```

```
class FuelTank {
  FuelType type;
  double amount;
  double costPerLiter;
}
```



“diesel”

Variable name	Value
diesel	
dieselTank	
dieselTank2	

Field name	Value
name	
fuelConsumption	0.3

Field name	Value
type	
amount	20
costPerLiter	1.35

Field name	Value
type	
amount	15
costPerLiter	1.35

```
FuelType diesel = new FuelType(“diesel“, 0.3);
FuelTank dieselTank = new FuelTank(diesel, 20, 1.3);
FuelTank dieselTank2 = new FuelTank(diesel, 15, 1.2);
```

Change the price of diesel to 1.35

```
dieselTank.costPerLiter = 1.35;
dieselTank2.costPerLiter = 1.35;
```

Complete example Discussion

- But what happens then if we do not have access to all diesel tanks?
- Object oriented programs should be designed to avoid redundancy
  - If the same information is contained in more than one place, you are wasting memory, and opening the door to inconsistencies
- It is therefore fundamental that each class
  - Has an (internal) state/data (fields)
  - Provides some functionalities (methods)
  - Represents a class of real world objects

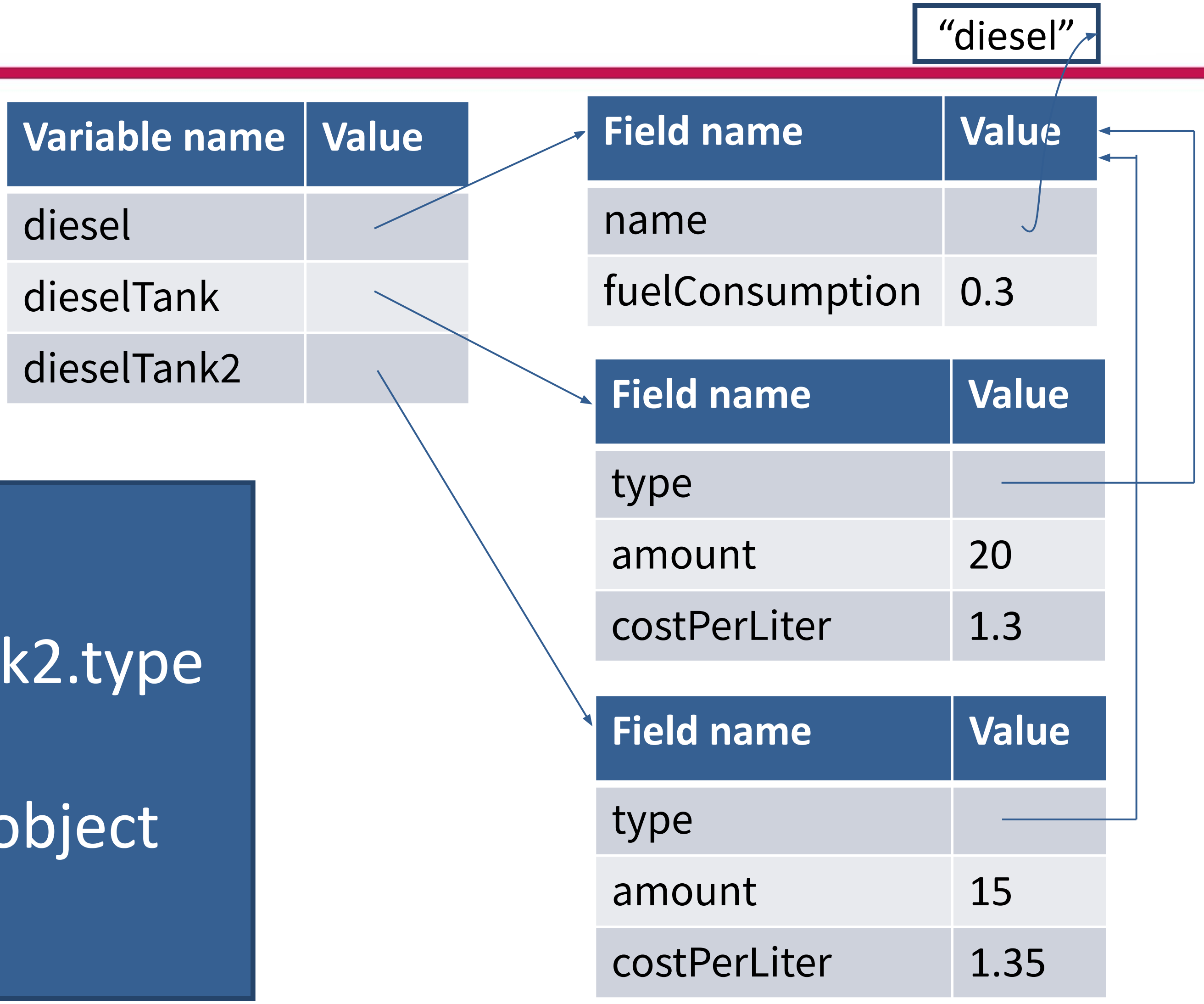
Aliasing

In computing, aliasing describes a situation in which a data location in memory can be accessed through different symbolic names in the program. Thus, modifying the data through one name implicitly modifies the values associated with all aliased names, which may not be expected by the programmer. As a result, aliasing makes it particularly difficult to understand, analyze and optimize programs.

WIKIPEDIA
The Free Encyclopedia

- Aliasing is a key feature of object-oriented programs
- It allows to share data among different software components
- However, it does not allow to locally reason to a piece of code

A complete example



Variable name	Value
diesel	
dieselTank	
dieselTank2	

Field name	Value
name	
fuelConsumption	0.3

Field name	Value
type	
amount	20
costPerLiter	1.3

Field name	Value
type	
amount	15
costPerLiter	1.35

- Diesel, dieselTank.type, dieselTank2.type are all aliases
- They are references to the same object

Packages

- Classes can be grouped in packages
- Each package represents a software unit
  - Distributable standalone
  - Combined with other units (aka, packages)
- For instance, a package about fuel
- package keyword at the beginning
- import before the declaration of the class to import other packages
  - * to import all classes in a package
- Naming convention: reverted Internet URLs
  - Avoid clashes

```
package it.unive.dais.po1.fuel;
class FuelTank { ...}
```

```
package it.unive.dais.po1.fuel;
class FuelType {...}
```

```
package it.unive.dais.po1.car;
import it.unive.dais.po1.fuel.*;
class Car {...}
```

Directory structure

- it.unive.dais.po1 should stay in directory it/unive/dais/po1
  - Otherwise it does not compile!
- The directory tree structure reflects the package hierarchy

Textbook

- Lecture notes: Chapter 3.3-3.8

- Arnold&others:
  – Packages:
    - Sections 18 (beginning of the chapter), 18.1, and 18.2
  – Variables, parameters, fields: section 7.3
  – Objects: section 2.4
  – New operator: 1.7.1