Lecture 6

Frame-based expert systems

- Introduction, or what is a frame?
- Frames as a knowledge representation technique
- Inference in frame-based expert systems
- Methods and demons
- Interaction of frames and rules
- Buy Smart: a frame-based expert system
- Summary

Introduction, or what is a frame?

A frame is a data structure with typical knowledge about a particular object or concept.

Frames were first proposed by **Marvin Minsky** in the 1970s.

Boarding pass frames

QANTAS BOARDING PASS	
Carrier:	*QANTAS AIRWAYS*
Name:	*MR N BLACK*
Flight:	*QF 612*
Date:	*29DEC*
Seat:	*23A*
From:	*HOBART*
To:	*MELBOURNE*
Boarding:	*0620*
Gate:	*2*

AIR NEW ZEALAND BOARDING PASS	
Carrier:	*AIR NEW ZEALAND*
Name:	*MRS J WHITE*
Flight:	*NZ 0198*
Date:	*23NOV*
Seat:	*27K*
From:	*MELBOURNE*
To:	*CHRISTCHURCH*
Boarding:	*1815*
Gate:	*4*

- Each frame has its own name and a set of **attributes** associated with it. *Name*, *weight*, *height* and *age* are slots in the frame *Person*. *Model*, *processor*, *memory* and *price* are slots in the frame *Computer*. Each attribute or slot has a value attached to it.
- Frames provide a natural way for the structured and concise representation of knowledge.
- A frame provides a means of organising knowledge in **slots** to describe various attributes and characteristics of the object.
- Frames are an application of **object-oriented programming** for expert systems.

- **Object-oriented programming** is a programming method that uses *objects* as a basis for analysis, design and implementation.
- In object-oriented programming, an **object** is defined as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. All objects have identity and are clearly distinguishable. *Michael Black*, *Audi 5000 Turbo*, *HP Pavilion P6555A* are examples of objects.

- An object combines both data structure and its behaviour in a single entity. This is in sharp contrast to conventional programming, in which data structure and the program behaviour have concealed or vague connections.
- When an object is created in an object-oriented programming language, we first assign a name to the object, then determine a set of attributes to describe the object's characteristics, and at last write procedures to specify the object's behaviour.
- A knowledge engineer refers to an object as a **frame** (the term, which has become the AI jargon).

Frames as a knowledge representation technique

- The concept of a frame is defined by a collection of **slots**. Each slot describes a particular attribute or operation of the frame.
- Slots are used to store values. A slot may contain a default value or a pointer to another frame, a set of rules or procedure by which the slot value is obtained.

Typical information included in a slot

- **Frame name.**
- **Relationship of the frame to the other frames.** The frame *HP Pavilion P6555A* might be a member of the class *Computer*, which in turn might belong to the class *Hardware*.
- **Slot value.** A slot value can be symbolic, numeric or Boolean. For example, the slot *Name* has symbolic values, and the slot *Age* numeric values. Slot values can be assigned when the frame is created or during a session with the expert system.

- **Default slot value.** The default value is taken to be true when no evidence to the contrary has been found. For example, a car frame might have four wheels and a chair frame four legs as default values in the corresponding slots.
- **Range of the slot value.** The range of the slot value determines whether a particular object complies with the stereotype requirements defined by the frame. For example, the cost of a computer might be specified between \$750 and \$1500.
- **Procedural information.** A slot can have a procedure attached to it, which is executed if the slot value is changed or needed.

- Frame-based expert systems also provide an extension to the slot-value structure through the application of *facets*.
- A **facet** is a means of providing extended knowledge about an attribute of a frame.
- Facets are used to establish the attribute value, control end-user queries, and tell the inference engine how to process the attribute.

What are the class and instances?

- The word *frame* often has a vague meaning. The frame may refer to a particular object, for example the computer *HP Pavilion P6555A*, or to a group of similar objects. To be more precise, we will use the *instance-frame* when referring to a particular object, and the *class-frame* when referring to a group of similar objects.
- A **class-frame** describes a group of objects with common attributes. *Animal*, *person*, *car* and *computer* are all class-frames.
- Each frame "knows" its class.

Computer class

CLASS:	*Desktop*		
[Str]	***Item Code***:		
[Str]	***Brand***:		
[Str]	***Model***:		
[Str]	***Processor***:		
[Str]	***Installed RAM***:		
[Str]	***Hard Drive***:		
[Str]	***Graphical Card***:		
[Str]	***Optical Drive***:	[Default]	← *DVD Burner*
[Str]	***USB Ports***:		
[Str]	***Keyboard***:	[Default]	← *104-key*
[Str]	***Warranty***:	[Default]	← *1 year*
[N]	***Cost***:		
[Str]	***Stock***:	[Initial]	← *In stock*

Computer instances

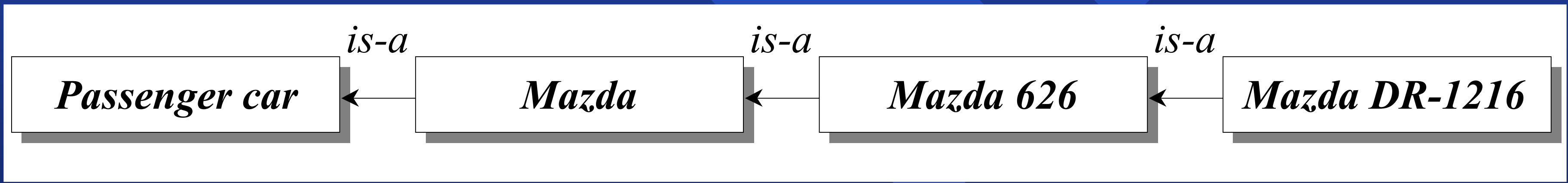
INSTANCE: *HP Pavilion P655A*		
Class:		*Desktop*
[Str]	***Item Code***:	*SY7973*
[Str]	***Brand***:	*HP*
[Str]	***Model***:	*Pavilion P6555A*
[Str]	***Processor***:	*Intel Core i5 3.2 GHz*
[Str]	***Installed RAM***:	*4 GB*
[Str]	***Hard Drive***:	*1 TB*
[Str]	***Graphical Card***:	*1 GB*
[Str]	***Optical Drive***:	*DVD Burner*
[Str]	***USB Ports***:	*8*
[Str]	***Keyboard***:	*104-key*
[Str]	***Warranty***:	*1 year*
[N]	***Cost***:	*1296.00*
[Str]	***Stock***:	*In stock*

INSTANCE: *Acer Aspire M3910*		
Class:		*Desktop*
[Str]	***Item Code***:	*SY7975*
[Str]	***Brand***:	*Acer*
[Str]	***Model***:	*Aspire M3910*
[Str]	***Processor***:	*Intel Core i7 2.8 GHz*
[Str]	***Installed RAM***:	*8 GB*
[Str]	***Hard Drive***:	*2 TB*
[Str]	***Graphical Card***:	*1 GB*
[Str]	***Optical Drive***:	*DVD Burner*
[Str]	***USB Ports***:	*10*
[Str]	***Keyboard***:	*104-key*
[Str]	***Warranty***:	*1 year*
[N]	***Cost***:	*2147.00*
[Str]	***Stock***:	*In stock*

Class inheritance in frame-based systems

- Frame-based systems support **class inheritance**.
- The fundamental idea of inheritance is that attributes of the class-frame represent things that are *typically* true for all objects in the class. However, slots in the instance-frames can be filled with actual data uniquely specified for each instance.

Relations of the car frames



Inheritance of slot values

CLASS:	*Passenger car*
[C]	***Engine type*** *In-line 4 cylinder*: *V6*:
[N]	***Horsepower*:**
[C]	***Drivetrain type*** *Rear wheel drive*: *Front wheel drive*: *Four wheel drive*:
[C]	***Transmission type*** *5-speed manual*: *4-speed automatic*:
[N]	***Fuel consumption (mpg)*:**
[N]	***Seating capacity*:**

CLASS:	*Mazda*
***Superclass*:**	*Passenger car*
[C]	***Engine type*** *In-line 4 cylinder*: *V6*:
[N]	***Horsepower*:**
[C]	***Drivetrain type*** *Rear wheel drive*: *Front wheel drive*: *Four wheel drive*:
[C]	***Transmission type*** *5-speed manual*: *4-speed automatic*:
[N]	***Fuel consumption (mpg)*:**
[N]	***Seating capacity*:**
[Str]	***Country of manufacture*:** *Japan*

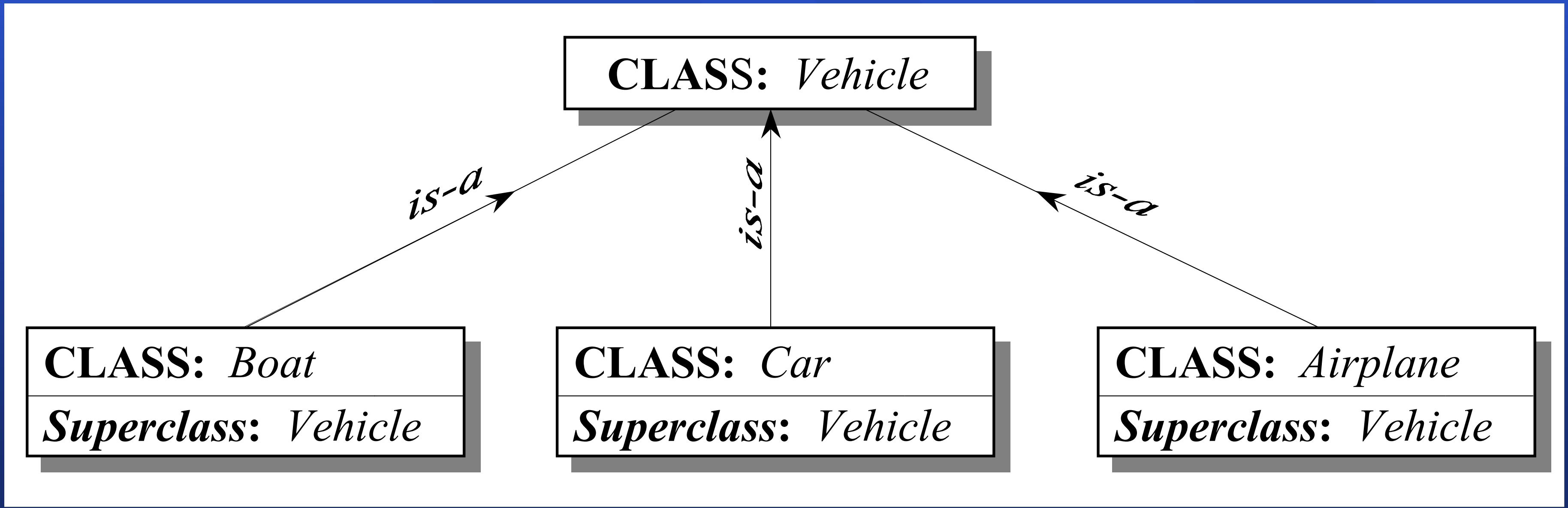
Inheritance of slot values (continued)

CLASS:	*Mazda 626*	
Superclass:	*Mazda*	
[C]	***Engine type***	
	In-line 4 cylinder:	
	V6:	
[N]	***Horsepower:***	*125*
[C]	***Drivetrain type***	
	Rear wheel drive:	
	Front wheel drive:	
	Four wheel drive:	
[C]	***Transmission type***	
	5-speed manual:	
	4-speed automatic:	
[N]	***Fuel consumption (mpg):***	*22*
[N]	***Seating capacity:***	*5*
[Str]	***Country of manufacture:***	*Japan*
[Str]	***Model:***	
[C]	***Colour***	
	Glacier White:	
	Sage Green Metallic:	
	Slate Blue Metallic:	
	Black Onyx Clearcoat:	
[Str]	***Owner:***	

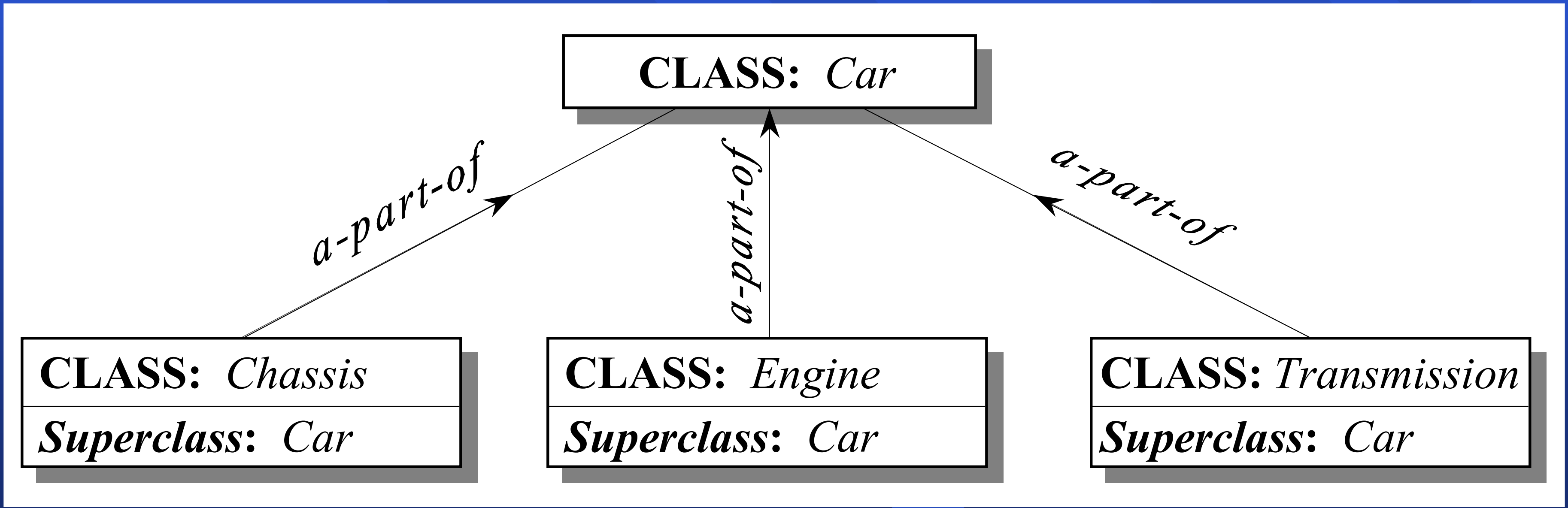
INSTANCE:	*Mazda DR-1216*	
Class:	*Mazda 626*	
[C]	***Engine type***	
	In-line 4 cylinder:	TRUE
	V6:	FALSE
[N]	***Horsepower:***	*125*
[C]	***Drivetrain type***	
	Rear wheel drive:	*FALSE*
	Front wheel drive:	TRUE
	Four wheel drive:	FALSE
[C]	***Transmission type***	
	5-speed manual:	FALSE
	4-speed automatic:	TRUE
[N]	***Fuel consumption (mpg):***	*28*
[N]	***Seating capacity:***	*5*
[Str]	***Country of manufacture:***	*Japan*
[Str]	***Model:***	*DX*
[C]	***Colour***	
	Glacier White:	FALSE
	Sage Green Metallic:	TRUE
	Slate Blue Metallic:	FALSE
	Black Onyx Clearcoat:	FALSE
[Str]	***Owner:***	*Mr Black*

Relationships among objects

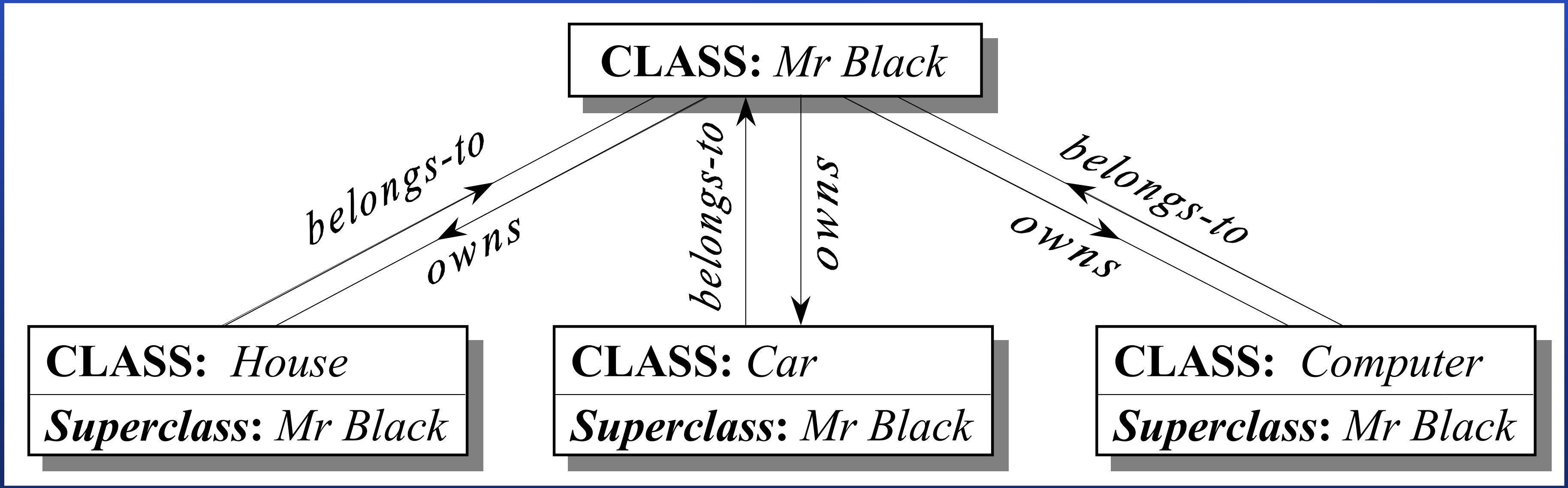
- **Generalisation** denotes ***a-kind-of*** or ***is-a*** relationship between superclass and its subclasses. For example, a car *is a* vehicle, or in other words, *Car* represents a subclass of the more general superclass *Vehicle*. Each subclass inherits all features of the superclass.



- **Aggregation** is ***a-part-of*** or ***part-whole*** relationship in which several subclasses representing *components* are associated with a superclass representing a *whole*. For example, an engine is *a part of* a car.



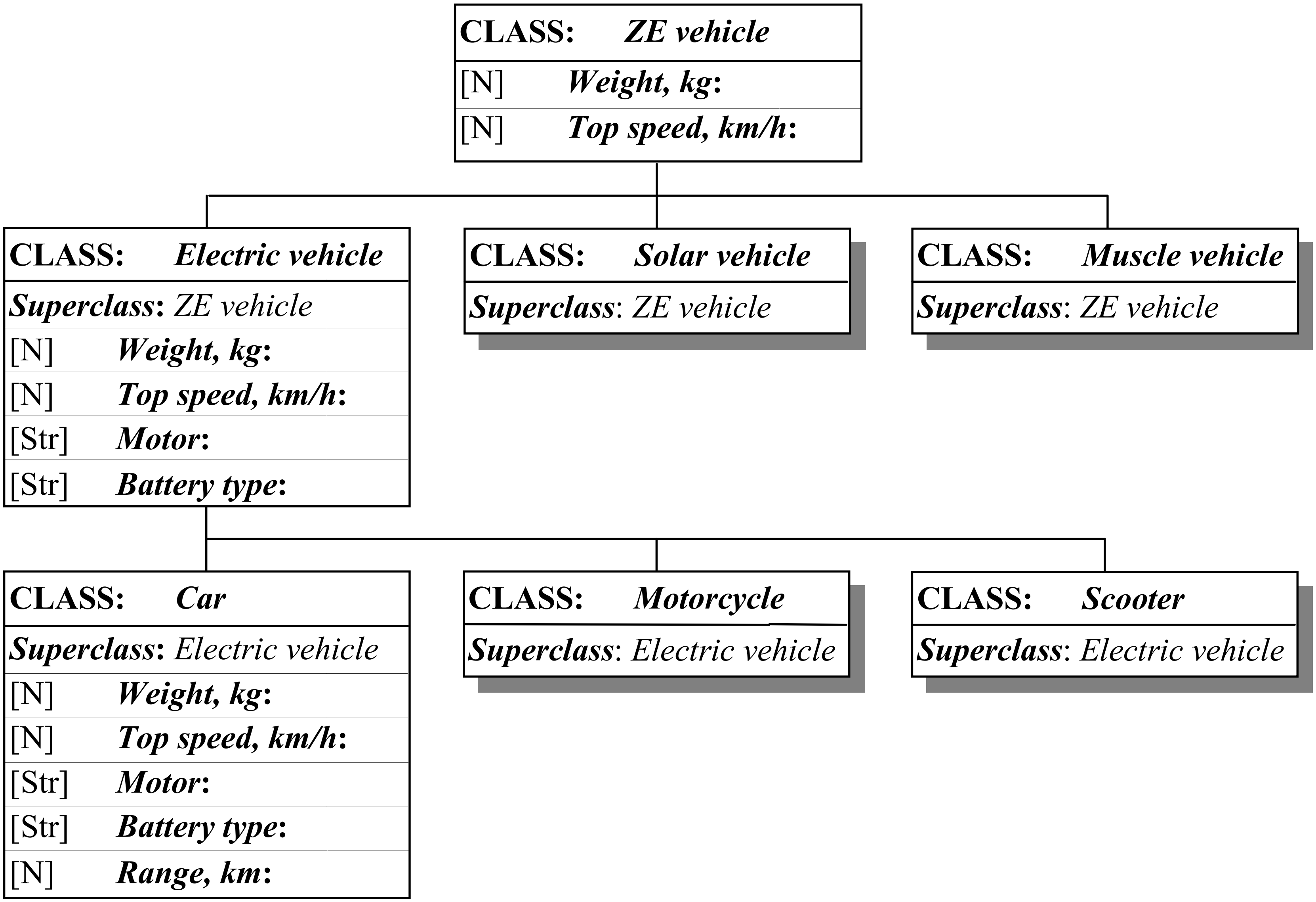
- **Association** describes some semantic relationship between different classes which are unrelated otherwise. For example, Mr Black owns a house, a car and a computer. Such classes as *House*, *Car* and *Computer* are mutually independent, but they are linked with the frame *Mr Black* through the semantic association.



Inheritance in frame-based systems

- Inheritance is defined as the process by which all characteristics of a class-frame are assumed by the instance-frame.
- A common use of inheritance is to impose default features on all instance-frames. We can create just one class-frame that contains generic characteristics of some object, and then obtain several instance-frames without encoding the class-level characteristics.

One-parent inheritance in zero-emission vehicles



One-parent inheritance (continued)

CLASS:	***Van***	
***Superclass*:** *Car*		
[N]	***Weight, kg*:**	
[N]	***Top speed, km/h*:**	
[Str]	***Motor*:**	
[Str]	***Battery type*:**	
[N]	***Range, km*:**	
[N]	***Payload, kg*:**	

INSTANCE:	***Ford Ecostar***	
***Class*:**		*Van*
[N]	***Weight, kg*:**	*1851*
[N]	***Top speed, km/h*:**	*113*
[Str]	***Motor*:**	*3-phase, AC induction*
[Str]	***Battery type*:**	*Sodium sulfur*
[N]	***Range, km*:**	*161*
[N]	***Payload, kg*:**	*463*

Multiple inheritance

CLASS:	*Electric vehicle*
[Str]	***Motor***:
[Str]	***Battery type***:

CLASS:	*Solar vehicle*
[Str]	***Solar panel type***:
[Str]	***Solar cell material***:

CLASS:	*Muscle vehicle*
[N]	***Number of wheels***:
[N]	***Pedalling manpower***:

CLASS:	*Muscle-Solar-Electric vehicle*
Superclass:	*Electric vehicle* *Solar vehicle* *Muscle vehicle*
[Str]	***Motor***:
[Str]	***Battery type***:
[Str]	***Solar panel type***:
[Str]	***Solar cell material***:
[N]	***Number of wheels***:
[N]	***Pedalling manpower***:
[N]	***Weight, kg***:
[N]	***Top speed, km/h***:

INSTANCE:	*Didik Muscle Car*	
Class:		*Muscle-Solar-Electric vehicle*
[Str]	***Motor***:	*24V DC*
[Str]	***Battery type***:	*Sealed lead acid*
[Str]	***Solar panel type***:	*BP140*
[Str]	***Solar cell material***:	*Crystalline silicon*
[N]	***Number of wheels***:	*4*
[N]	***Pedalling manpower***:	*2*
[N]	***Weight, kg***:	*60*
[N]	***Top speed, km/h***:	*35*

Methods and demons

Expert systems are required not only to store the knowledge but also to validate and manipulate this knowledge. To add actions to our frames, we need **methods** and **demons**.

- A method is a procedure associated with a frame attribute that is executed whenever requested.
- We write a method for a specific attribute to determine the attribute's value or execute a series of actions when the attribute's value changes.
- Most frame-based expert systems use two types of methods:

  WHEN CHANGED and **WHEN NEEDED**.

What is the difference between methods and demons?

- A demon has an IF-THEN structure. It is executed whenever an attribute in the demon's IF statement changes its value. In this sense, demons and methods are very similar, and the two terms are often used as synonyms.
- However, methods are more appropriate if we need to write complex procedures. Demons, on the other hand, are usually limited to IF-THEN statements.

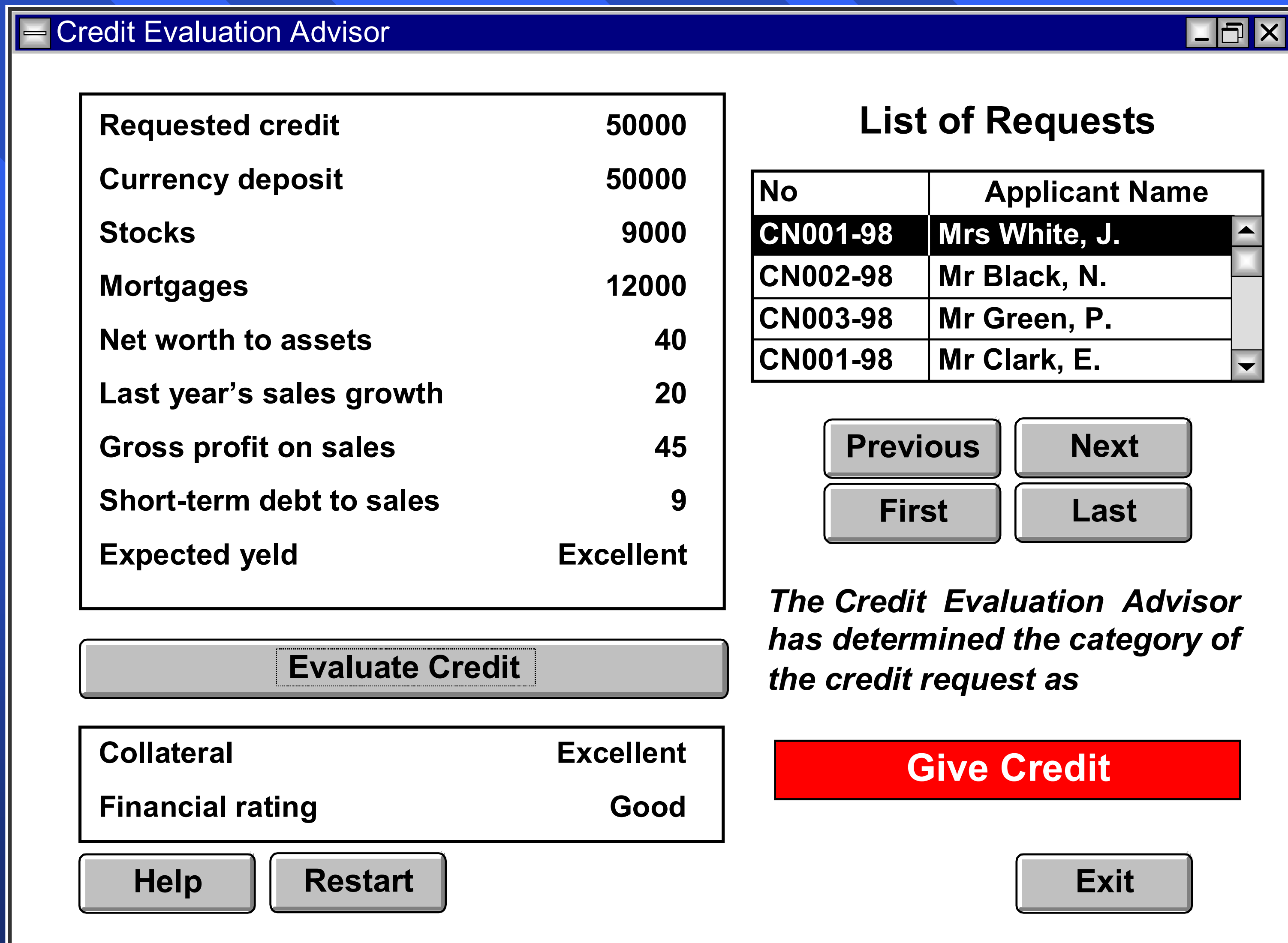
WHEN CHANGED method

A WHEN CHANGED method is executed immediately when the value of its attribute changes.

To understand how WHEN CHANGED methods work, we consider a simple problem.

- The expert system is required to assist a loan officer in evaluating credit requests from small business ventures.
- A credit request is to be classified into one of three categories, "Give credit", "Deny credit" or "Consult a superior".
- When a loan officer provides a qualitative rating of the expected yield from the loan, the expert system compares the business collateral with the amount of credit requested, evaluates a financial rating based on a weighted sum of the business's net worth to assets, last year's sales growth, gross profit on sales and short-term debt to sales, and determines a category for the credit request.

Input display for the request selection



The class *Action Data* and WHEN CHANGED methods

CLASS:	***Action Data***
[S] ***Goto Next***:	[WHEN CHANGED]
[S] ***Goto Previous***:	[WHEN CHANGED]
[S] ***Goto First***:	[WHEN CHANGED]
[S] ***Goto Last***:	[WHEN CHANGED]

INSTANCE:	**Action Data 1**
Class:	*Action Data*
[S] ***Goto Next***:	TRUE
[S] ***Goto Previous***:	UNDETERMINED
[S] ***Goto First***:	UNDETERMINED
[S] ***Goto Last***:	UNDETERMINED

```
WHEN CHANGED
 BEGIN
  Current Request Number := Request Number OF Request
  FIND Request
   LIMIT 1
   WHERE  Request Number OF Request > Current Request Number
  FIND END
 END
```

```
WHEN CHANGED
 BEGIN
  Target Request Number := Request Number OF Request - 1
  IF Target Request Number > 0 THEN
    BEGIN
      FIND Request
        LIMIT 1
     WHERE Request Number OF Request = Target Request Number
     FIND END
    END
END
```

```
WHEN CHANGED
  BEGIN
    FIND Request
      LIMIT 1
    FIND END
  END
```

```
WHEN CHANGED
  BEGIN
   Target Request Number := Total Number of Requests
   FIND Request
     LIMIT 1
     WHERE  Request Number OF Request = Target Request Number
   FIND END
 END
```

Class *Request*

CLASS: *Request*
[Str] ***Applicant's name*:**
[Str] ***Application no.*:**
[N] ***Requested credit*:**
[N] ***Currency deposits*:**
[N] ***Stocks*:**
[N] ***Mortgages*:**
[N] ***Net worth to assets*:**
[N] ***Last year's sales growth*:**
[N] ***Gross profits on sales*:**
[N] ***Short-term debt to sales*:**
[C] ***Expected yield*** *Excellent*: *Reasonable*: *Poor*:
[N] ***Request Number*:**

Instances of the Class *Request*

INSTANCE:	Request 1
Class:	*Request*
[Str] ***Applicant's name***:	*Mrs White, J.*
[Str] ***Application no.***:	CN001-98
[N] ***Requested credit***:	50000
[N] ***Currency deposits***:	50000
[N] ***Stocks***:	9000
[N] ***Mortgages***:	12000
[N] ***Net worth to assets***:	40
[N] ***Last year's sales growth***:	20
[N] ***Gross profits on sales***:	45
[N] ***Short-term debt to sales***:	9
[C] ***Expected yield***	
Excellent:	TRUE
Reasonable:	FALSE
Poor:	FALSE
[N] ***Request Number***:	1

INSTANCE:	Request 2
Class:	*Request*
[Str] ***Applicant's name***:	*Mr Black, N.*
[Str] ***Application no.***:	CN002-98
[N] ***Requested credit***:	75000
[N] ***Currency deposits***:	45000
[N] ***Stocks***:	10000
[N] ***Mortgages***:	20000
[N] ***Net worth to assets***:	45
[N] ***Last year's sales growth***:	25
[N] ***Gross profits on sales***:	35
[N] ***Short-term debt to sales***:	10
[C] ***Expected yield***	
Excellent:	FALSE
Reasonable:	TRUE
Poor:	FALSE
[N] ***Request Number***:	2

WHEN NEEDED method

A WHEN NEEDED method is used to obtain the attribute value only when it is needed.

A WHEN NEEDED method is executed when information associated with a particular attribute is needed for solving the problem, but the attribute value is undetermined.

Interaction of frames and rules

Most frame-based expert systems allow us to use a set of rules to evaluate information contained in frames.

How does an inference engine work in a frame based system?

- In a rule-based expert system, the inference engine links the rules contained in the knowledge base with data given in the database.
- When the goal is set up, the inference engine searches the knowledge base to find a rule that has the goal in its consequent.
- If such a rule is found and its IF part matches data in the database, the rule is fired and the specified object, the goal, obtains its value. If no rules are found that can derive a value for the goal, the system queries the user to supply that value.

- In a frame-based system, the inference engine also searches for the goal.

But:

- In a frame-based system, rules play an auxiliary role. Frames represent here a major source of knowledge, and both methods and demons are used to add actions to the frames.
- Thus, the goal in a frame-based system can be established either in a method or in a demon.

Example:

Suppose we want to evaluate the credit request selected by the user. The expert system is expected to begin the evaluation when the user clicks the *Evaluate Credit* pushbutton on the input display. This pushbutton is attached to the attribute *Evaluate Credit* of the class *Credit Evaluation*.

The *Credit Evaluation* class, WHEN CHANGED and WHEN NEEDED methods

CLASS:	***Credit Evaluation***	
[S] ***Evaluate Credit*:**		[WHEN CHANGED]
[C] ***Collateral*:**		
	Excellent:	
	Good:	
	Moderate:	
[C] ***Financial rating*:**		
	Excellent:	
	Good:	
	Medium:	
	Bad:	
[C] ***Evaluation*:**		[WHEN NEEDED]
	Give credit:	
	Deny credit:	
	Consult a superior:	

```
WHEN CHANGED
 BEGIN
  PURSUE Evaluation OF Credit Evaluation
 END
```

```
WHEN NEEDED
 BEGIN
  Evaluation OF Credit Evaluation IS Consult
  superior := TRUE
 END
```

Example of rules for credit evaluation

RULE 1
IF Currency deposits OF Request >= Requested credit OF Request
THEN Collateral OF Credit Evaluation IS Excellent

RULE 2
IF Currency deposits OF Request >= Requested credit OF Request * 0.7
AND (Currency deposits OF Request + Stocks OF Request) >= Requested credit OF Request
THEN Collateral OF Credit Evaluation IS Excellent

RULE 3
IF (Currency deposits OF Request + Stocks OF Request) > Requested credit OF Request * 0.6
AND (Currency deposits OF Request + Stocks OF Request) < Requested credit OF Request * 0.7
AND (Currency deposits OF Request + Stocks OF Request + Mortgages OF Request) >= Requested credit OF Request
THEN Collateral OF Credit Evaluation IS Good

RULE 4
IF (Currency deposits OF Request + Stocks OF Request + Mortgages OF Request) <= Requested credit OF Request
THEN Collateral OF Credit Evaluation IS Moderate

RULE 5
IF Net worth to assets OF Request * 5 + Last year's sales growth OF Request + Gross profits on sales OF Request * 5 + Short term debt to sales OF Request * 2 <= -500
THEN Financial rating OF Credit Evaluation IS Bad

- Based on the set of rules provided for credit evaluation, the inference engine cannot establish the value of the attribute *Evaluation* in some cases.
- We can use the **WHEN NEEDED method** to establish the attribute value.
- The WHEN NEEDED method is attached to the attribute *Evaluation*. The inference engine executes this method when it needs to determine the value of *Evaluation*. When the WHEN NEEDED method is executed, the attribute *Evaluation* receives the value *Consult a superior*.

Buy Smart: a Frame-based Expert System

The development of a frame-based system typically involves the following steps:

1. Specify the problem and define the scope of the system.
2. Determine classes and their attributes.
3. Define instances.
4. Design displays.
5. Define WHEN CHANGED and WHEN NEEDED methods, and demons.
6. Define rules.
7. Evaluate and expand the system.

Step 1:

Specify the problem and define the scope of the system

We start by collecting some information about properties for sale in our region. We can identify relevant details such as the property type, location, number of bedrooms and bathrooms, and of course, the property price. We also should provide a short description and a nice photo for each property.

Step 1: (Continued)

The next step is to list all possible queries we might think of:

- What is the maximum amount you want to spend on a property?
- What type of the property do you prefer?
- Which suburb would you like to live in?
- How many bedrooms do you want?
- How many bathrooms do you want?

Step 2:
Determine classes and their attributes

- We begin with the general or conceptual type of classes. For example, we can talk about the concept of a *property* and describe general features that are common to most properties. We can characterise each property by its location, price, type, number of bedrooms and bathrooms, construction, picture and description.

- We also need to present contact details of the property, such as its address or phone number.

Class *Property*

CLASS: ***Property***
[Str] ***Area***:
[Str] ***Suburb***:
[N] ***Price***:
[Str] ***Type***:
[N] ***Bedrooms***:
[N] ***Bathrooms***:
[Str] ***Construction***:
[Str] ***Phone***:
[Str] ***Pictfile***:
[Str] ***Textfile***:
[N] ***Instance Number***:

Step 3:
Define instances

- Once we determined the class-frame *Property*, we can create its instances by using data stored in the database.

- For most frame-based expert systems, this task requires us to tell the system that we want a new instance to be created.

Step 3: (Continued)

For example, to create a new instance of the class *Property* in *Level5 Object*, we use the following code:

```
MAKE Property
        WITH Area :=  area OF dB3 HOUSE 1
        WITH Suburb :=  suburb OF dB3 HOUSE 1
        WITH Price :=  price OF dB3 HOUSE 1
        WITH Type :=  type OF dB3 HOUSE 1
        WITH Bedrooms :=  bedrooms OF dB3 HOUSE 1
        WITH Bathrooms :=  bathrooms OF dB3 HOUSE 1
        WITH Construction :=  construct OF dB3 HOUSE 1
        WITH Phone :=  phone OF dB3 HOUSE 1
        WITH Pictfile :=  pictfile OF dB3 HOUSE 1
        WITH Textfile :=  textfile OF dB3 HOUSE 1
        WITH Instance Number :=  Current Instance Number
```

Instances of the Class *Property*

INSTANCE:	**Property 1**
Class:	*Property*
[Str] ***Area***:	*Central Suburbs*
[Str] ***Suburb***:	*New Town*
[N] ***Price***:	164000
[Str] ***Type***:	*House*
[N] ***Bedrooms***:	3
[N] ***Bathrooms***:	1
[Str] ***Construction***:	*Weatherboard*
[Str] ***Phone***:	(03) 6226 4212
[Str] ***Pictfile***:	*house01.bmp*
[Str] ***Textfile***:	*house01.txt*
[N] ***Instance Number***:	1

INSTANCE:	**Property 2**
Class:	*Property*
[Str] ***Area***:	*Central Suburbs*
[Str] ***Suburb***:	*Taroona*
[N] ***Price***:	150000
[Str] ***Type***:	*House*
[N] ***Bedrooms***:	3
[N] ***Bathrooms***:	1
[Str] ***Construction***:	*Brick*
[Str] ***Phone***:	(03) 6226 1416
[Str] ***Pictfile***:	*house02.bmp*
[Str] ***Textfile***:	*house02.txt*
[N] ***Instance Number***:	2

Step 4:
Design displays

- We need the *Application Title Display* to present some general information to the user at the beginning of each application. This display may consist of the application title, general description of the problem, representative graphics and also copyright information.

The *Application Title Display*

Buy Smart

Buy Smart

Buying any property – especially your first – is very exciting, but it is often a little nerve-racking as well. It is important you take the time to do your home work properly. This expert system will assists you in choosing a home to meet your particular requirements.

Exit

Continue

Step 4: (Continued)

- The next display is the *Query Display*. This display should allow us to indicate our preferences by answering the queries presented by the expert system.

The *Query Display*

Buy Smart

Select the most important things you are looking for in your property.

Suburb

- All Suburbs
- Central Suburbs
- Eastern Shore
- Northern Suburbs
- Southern Suburbs

Bedrooms

- Any number of bedrooms
- One bedroom
- Two bedrooms
- Three bedrooms
- Four bedrooms or more

Maximum Price

- No maximum
- $ 50,000
- $ 100,000
- $ 150,000
- $ 200,000
- $ 250,000
- $ 300,000
- $ 350,000
- $ 400,000
- $ 500,000
- $ 1,000,000
- $ 2,000,000

Property Type

- All property types
- House
- Unit
- Townhouse

Bathrooms

- Any number of bathrooms
- One bathroom
- Two bathrooms
- Three bathrooms or more

Help | Restart | Search

Step 4: (Continued)

- And finally, we should design the *Property Information Display*. This display has to provide us with the list of suitable properties, an opportunity to move to the next, previous, first or last property in the list, and also a chance to look at the property picture and its description.

The *Property Information Display*

Buy Smart

Individual Property

MODERNISED WITH ARCHITECT'S FLAIR
yet retaining the original charm. Three large bedrooms. Formal dining warmed by a wood heater, family room next to an impressive kitchen and overlooking lavish gardens. A sparkling bathroom and large laundry. Convenient location, garage and parking.

Search: List of Properties

12 *properties found*

Suburb	Price	Type
New Town	164000	House
Taroona	150000	House
North Hobart	127000	House
Taroona	110000	House
Lenah Valley	145000	House
South Hobart	140000	House
South Hobart	115000	House

Construction:	Weatherboard
Number of bedrooms:	3
Number of bathrooms:	1
Telephone:	(03) 6226 4212

Previous Next First Last

Help Restart Exit

Step 5:
Define WHEN CHANGED and WHEN NEEDED methods, and demons

- We must develop a way to bring our application to life. There are two ways to accomplish this task.
- The first one relies on WHEN CHANGED and WHEN NEEDED methods, and demons.
- The second approach involves pattern-matching rules. In frame-based systems, we always first consider an application of methods and demons.

Step 5: (Continued)

- We create all instances of the class *Property* at once when the user clicks on the *Continue* pushbutton on the *Application Title Display*, and then remove inappropriate instances step-by-step based on the user's preferences when he or she selects pushbuttons on the *Query Display*.

The WHEN CHANGED method of the attribute *Load Property*

CLASS:	***Action Data***
[S] ***Load Properties***:	[WHEN CHANGED]

INSTANCE:	**Action Data 1**
Class:	*Action Data*
[S] *Load Properties*:	TRUE

```
WHEN CHANGED
 BEGIN
 Current Instance Number := 0
 FORGET Property
 FIND dB3 HOUSE 1
 WHEN FOUND
  Current Instance Number := Current Instance Number + 1
  MAKE Property
       WITH Area :=  area OF dB3 HOUSE 1
       WITH Suburb :=  suburb OF dB3 HOUSE 1
       WITH Price :=  price OF dB3 HOUSE 1
       WITH Type :=  type OF dB3 HOUSE 1
       WITH Bedrooms :=  bedrooms OF dB3 HOUSE 1
       WITH Bathrooms :=  bathrooms OF dB3 HOUSE 1
       WITH Construction :=  construct OF dB3 HOUSE 1
       WITH Phone :=  phone OF dB3 HOUSE 1
       WITH Pictfile :=  pictfile OF dB3 HOUSE 1
       WITH Textfile :=  textfile OF dB3 HOUSE 1
       WITH Instance Number :=  Current Instance Number
 FIND END
 Total Number of Instances := Current Instance Number
 Goto First Property OF Action Data := TRUE
END
```

Demons for the *Query Display*

```
DEMON 1
IF selected OF Central Suburbs pushbutton
THEN FIND Property
     WHERE  Area OF Property <> "Central Suburbs"
     WHEN FOUND
       FORGET  CURRENT Property
   FIND END
          •
          •
          •
DEMON 5
IF selected OF House pushbutton
THEN FIND Property
     WHERE  Type OF Property <> "House"
     WHEN FOUND
       FORGET  CURRENT Property
   FIND END
```

The WHEN CHANGED method of the attributes *Load Instance Number* and *Goto First Property*

CLASS:	***Action Data***
[S] ***Load Properties***:	[WHEN CHANGED]
[S] ***Load Instance Number***:	[WHEN CHANGED]
[S] ***Goto First Property***:	[WHEN CHANGED]

INSTANCE:	**Action Data 1**
Class:	*Action Data*
[S] ***Load Properties***:	TRUE
[S] ***Load Instance Number***:	TRUE
[S] ***Goto First Property***:	TRUE

```
WHEN CHANGED
 BEGIN
   Current Instance Number := 0
   FIND Property
     WHEN FOUND
       Current Instance Number := Current Instance Number + 1
       Instance Number OF Property := Current Instance Number
   FIND END
   Total Number of Instances := Current Instance Number
   Goto First Property OF Action Data := TRUE
 END
```

```
WHEN CHANGED
 BEGIN
   FIND Property
     LIMIT 1
     WHEN FOUND
       filename OF Property picturebox := Pictfile OF Property
       filename OF Property textbox := Textfile OF Property
   FIND END
 END
```

Step 6:
Define rules

- When we design a frame-based expert system, one of the most important and difficult decisions is whether to use rules or manage with methods and demons instead. This decision is usually based on the personal preferences of the designer.
- In our application, we use methods and demons because they offer us a powerful but simple way of representing procedures.

<u>Step 7</u>:
Evaluate and expand the system

We have now completed the initial design of our *Buy Smart* expert system. The next task is to evaluate it. We want to make sure that the system's performance meets our expectations.