

COOL

Clips Object Oriented Language

Chapter 8 Matters of Inheritance

The easiest way to obtain wealth is to inherit it; the second best way is to make it off the labor of others; marrying wealth is too much like work.

This chapter is an overview of object-oriented programming in CLIPS. Unlike rule-based programming in which you can just jump right in and write a rule without caring what else is in the system, object-oriented programming requires some essential background material.

Paradigme programiranja

- Proceduralna
- Deklarativna
- Objektno orijentisana
- Bazirana na pravilima
- Neuralne mreže – konekcionistička
- Najvažniji zahtev danas je naravno što veća fleksibilnost softvera, uz podrazumevanu egzaktnost, funkcionalnost, performanse...
- CLIPS – pravila, OOP i proceduralna

CLIPS klase

- Pojam OOP-a u CLIPS-u je isti kao i u drugim OOP jezicima – klasičnim kao što su C++, Java, C#, itd.
- Pojam klase u CLIPS-u je isti kao i pojam klase u pomenutim jezicima.
- Klasa je apstrakcija tipova entiteta iz realnog sveta koji treba softverski podržati i istovremeno predstavlja šablon – template na osnovu koga se kreiraju objekti
- Apstrakcija znači da se pažnja posvećuje samo bitnim karakteristikama objekata iz realnog sveta – atributi i metode, dok se ostali manje važni zanemaruju – ne prikazuju uopšte

Osnovne karakteristike OOP

- Apstrakcija
 - Enkapsulacija
 - Nasleđivanje
 - Polimorfizam
 - Dinamičko vezivanje
-
- Klasa se sastoji iz atributa i metoda
 - Atributi se nazivaju i svojstva ili imenovani slotovi koji opisuju klasu
 - Metodi se nazivaju message handlers ili handlers i opisuju ponašanje klase - objekata

Nasleđivanje

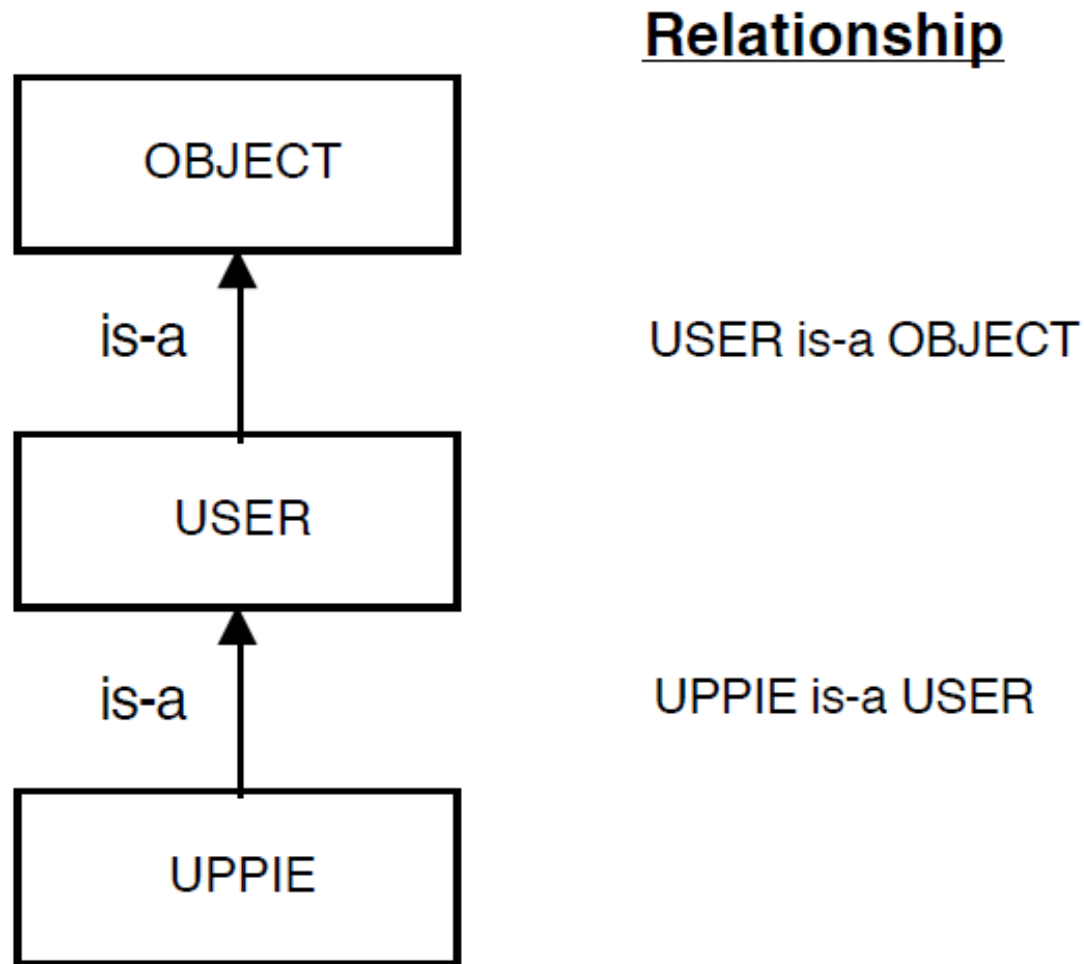


Fig. 1.1 The UPPIE Class

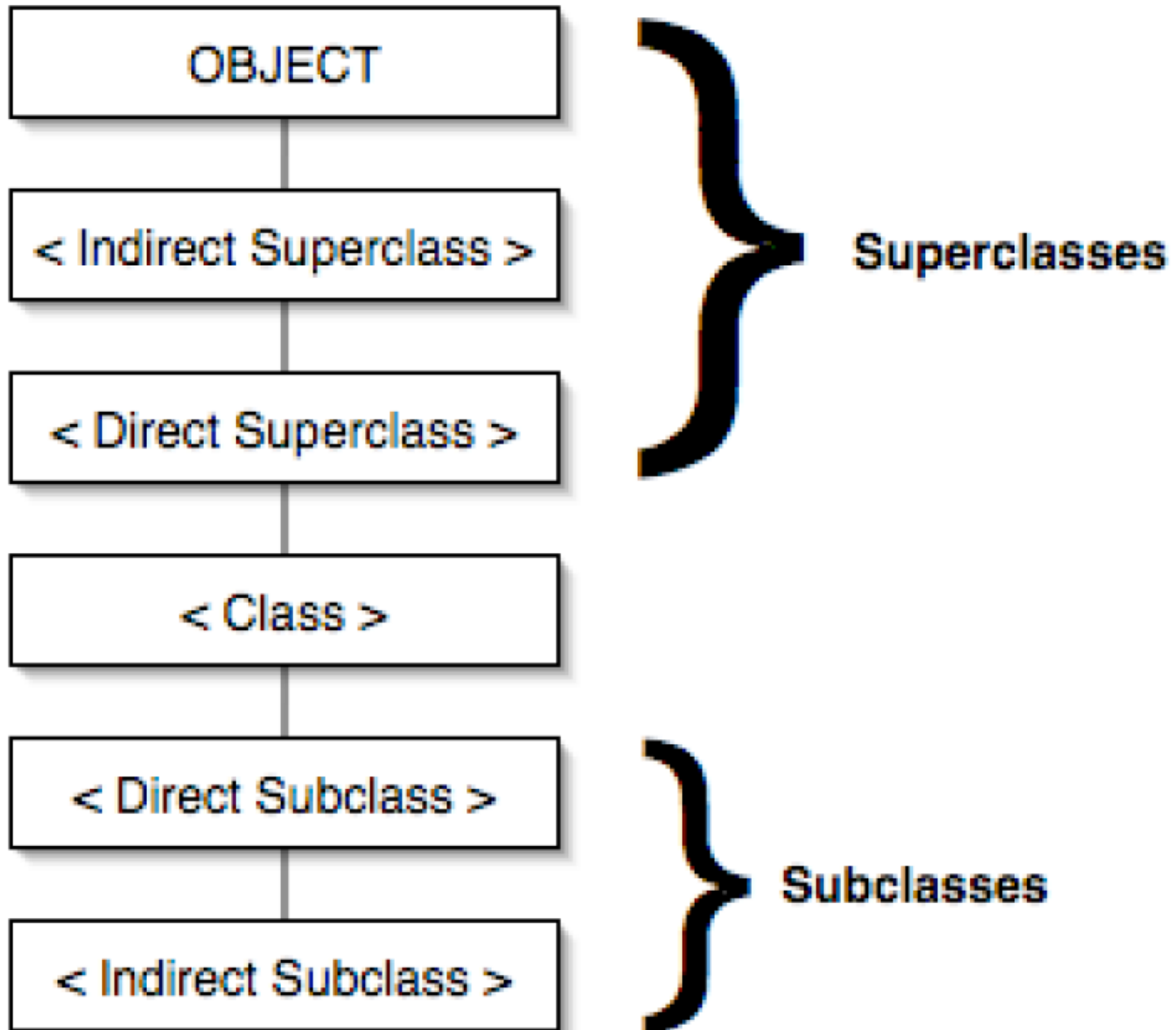
Nasleđivanje

- Nasleđivanje se koristi za kreiranje novih klasa koje imaju zajedničke karakteristike sa već postojećim klasama
- Nasleđivanjem nova klasa preuzima već postojeće članove roditeljske klase
- Time se štedi na nepotrebnom ponovnom definisanju nečega što već postoji, kao i na testiranju, pošto se pretpostavlja da je stara klasa već detaljno istestirana i samim tim potpuno proverena
- Odnos nasleđivanja između klasa je tipa is-a u prevodu **je**, klasa naslednik je tipa klase od koje nasleđuje i predstavlja podtip te klase

Defclass konstrukcija

- (defclass UPPIE (is-a USER))
- Klasa USER je već definisana u CLIPS-u, tako da se ne može ponovo definisati
- Klasa USER nasleđuje od klase OBJECT koja je osnovna klasa u CLIPS-u i naravno predefinisana pa se ne može ponovo kreirati
- Svaka korisnička klasa u CLIPS-u mora da nasleđuje od neke klase koja je već definisana u CLIPS-u
- (defclass <class> (is-a <direct-superclasses>))
- **direct superclass precedence list – redosled nasleđivanja - superklasa**
- (defclass DUCK (is-a DUCKLING USER OBJECT))

Klasse i podklase



Nasleđivanje

- Klasa OBJECT u CLIPS-u je osnovna klasa u CLIPS-u koja nema klasu od koje nasleđuje, to je ROOT klasa
- Osovni princip OOP:
- Klasa može da nasledi od svih klasa duž putanje nasleđivanja, tj. od svih svojih superklasa
- Nasleđeni slotovi i handler-i se mogu redefinisati u podklasama
- **inheritance path** je putanja nasleđivanja koju sačinjava niz povezanih klasa u odnosu nasleđivanja sve do klase OBJECT

Nasleđivanje opisuje taxonomy – taksonomiju ili klasifikaciju

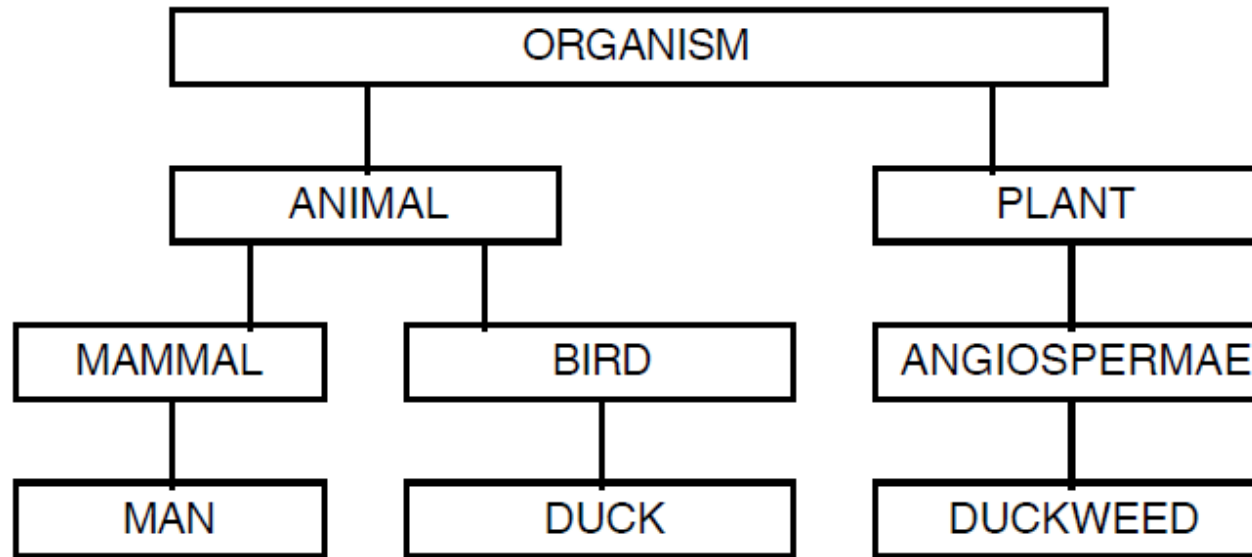


Fig. 1.3 Simple Taxonomy of Living Organisms With is-a Links

- Putanje nasleđivanja u biologiji
- Klasifikacija u biologiji naglašava sličnosti između organizama i grupiše ih u srodne celine

Putanje nasleđivanja tipa drveta

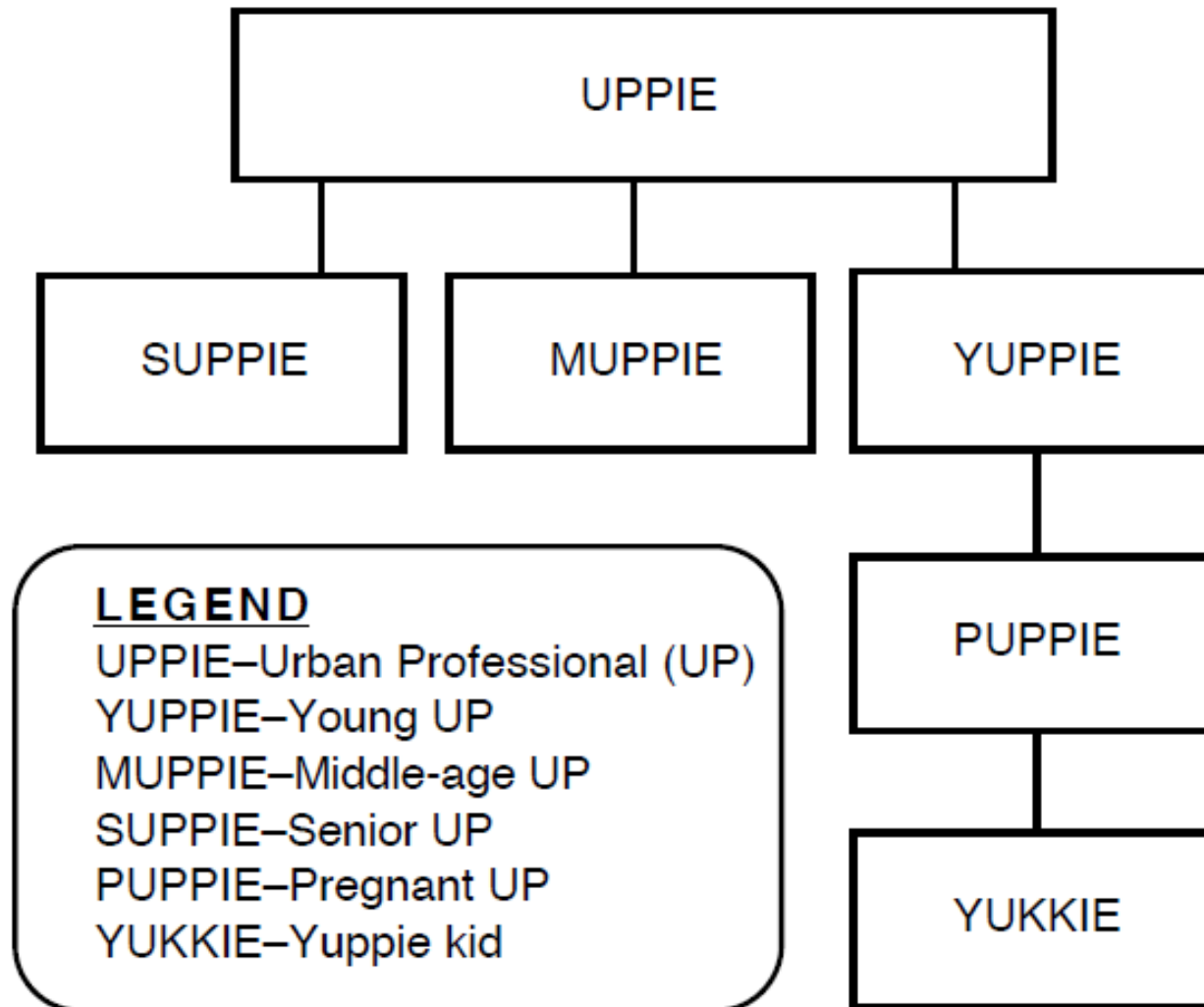


Fig. 1.5 The Illegitimate YUKKIE

Višestruko nasleđivanje

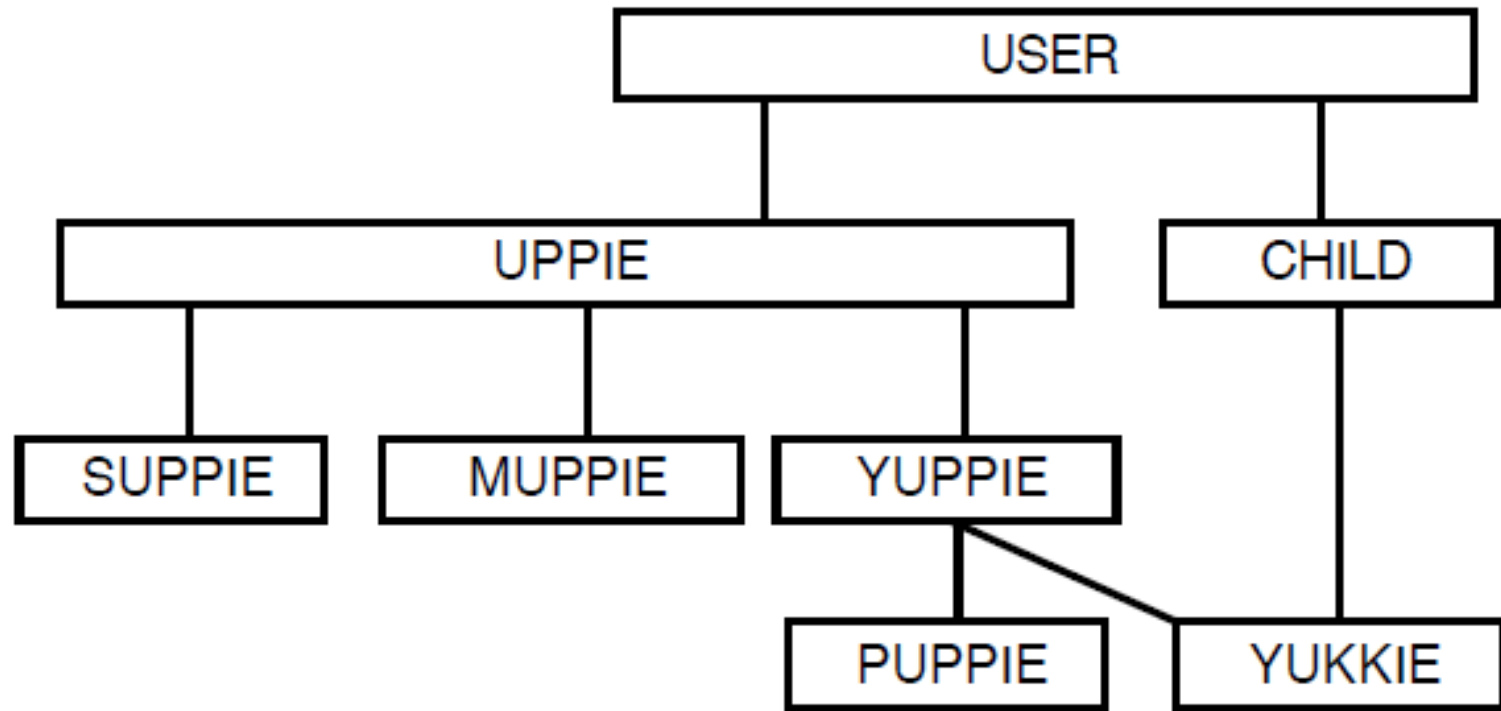


Fig. 1.6 The Legitimate YUKKIE

- Tačan opis nasleđivanja preko višestrukog nasleđivanja koje je nedostupno u nekim OOP jezicima – JAVA, C#

Odnosi nasleđivanja između klasa

- CLIPS> (clear)
- CLIPS> (defclass UPPIE (is-a USER))
- CLIPS> (defclass CHILD (is-a USER))
- CLIPS> (defclass SUPPIE (is-a UPPIE))
- CLIPS> (defclass MUPPIE (is-a UPPIE))
- CLIPS> (defclass YUPPIE (is-a UPPIE))
- CLIPS> (defclass PUPPIE (is-a YUPPIE))
- CLIPS> (defclass YUKKIE (is-a YUPPIE CHILD))
- Svaka klasa mora da bude definisana pre svoje podklase, u suprotnom se dobija poruka o grešci

Funkcije za klase u CLIPS-u

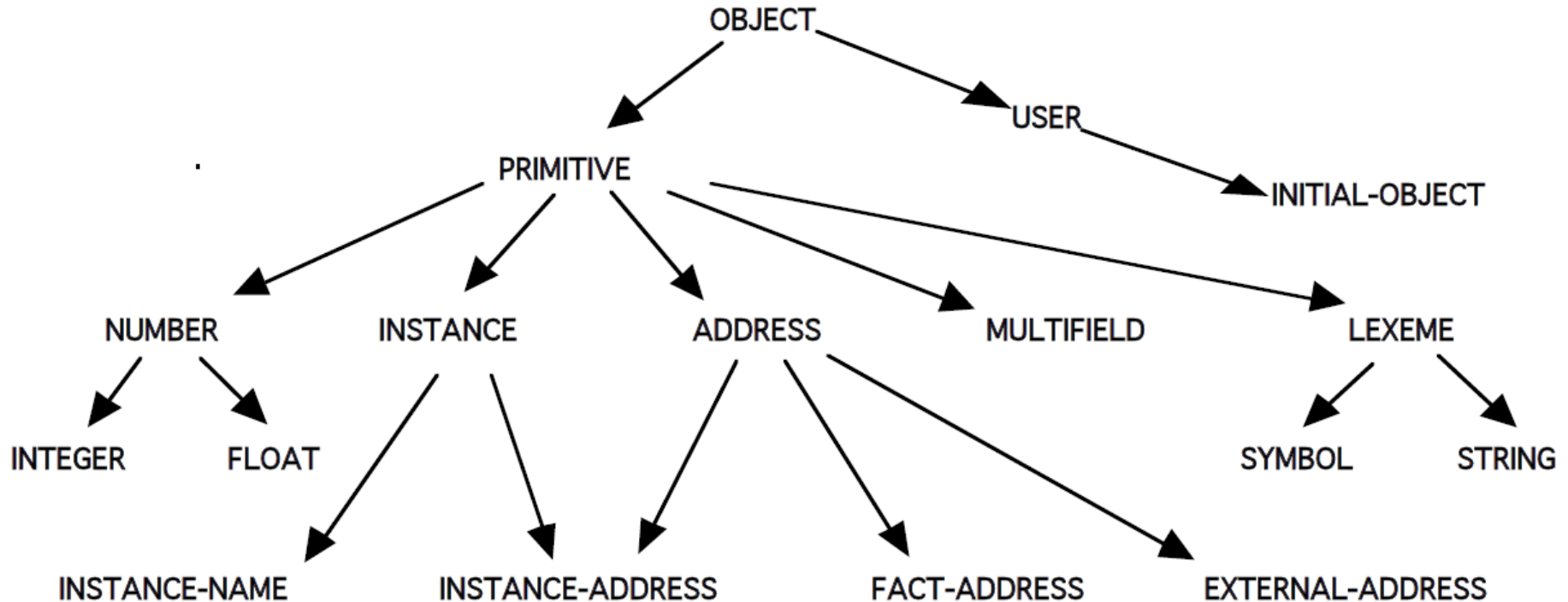
- (function <class1> <class2>)
- CLIPS> (superclassp UPPIE YUPPIE)
- TRUE
- CLIPS> (superclassp YUPPIE UPPIE)
- FALSE
- CLIPS> (subclassp YUPPIE UPPIE)
- TRUE
- CLIPS> (subclassp UPPIE YUPPIE)
- FALSE
- CLIPS>

list-defclasses command

```
CLIPS> (list-defclasses)
FLOAT                ADDRESS
INTEGER              INSTANCE
SYMBOL               USER
STRING               INITIAL-OBJECT
MULTIFIELD            UPPIE
EXTERNAL-ADDRESS      CHILD
FACT-ADDRESS          SUPPIE
INSTANCE-ADDRESS      MUPPIE
INSTANCE-NAME         YUPPIE
OBJECT                PUPPIE
PRIMITIVE             YUKKIE
NUMBER                For a total of 24 defclasses.
LEXEME                CLIPS>
```

- Izlistane postojeće klase u CLIPS-u

Predefinisane klase u CLIPS-u



- Default objekat – [initial-object] koji se kreira posle reset ili clear, je objekat klase INITIAL-OBJECT koja nasleđuje od klase USER. Preporuka je da korisnički objekti nasleđuju od klase USER, jer ova klasa ima najviše korisnih message-handler-a.

The **browse-classes** command shows the class hierarchy through indentation

```
CLIPS> (browse-classes)
```

```
OBJECT
```

```
  PRIMITIVE
```

```
    NUMBER
```

```
      INTEGER
```

```
      FLOAT
```

```
    LEXEME
```

```
      SYMBOL
```

```
      STRING
```

```
  MULTIFIELD
```

```
  ADDRESS
```

```
    EXTERNAL-ADDRESS
```

```
    FACT-ADDRESS
```

```
    INSTANCE-ADDRESS *
```

```
  INSTANCE
```

```
    INSTANCE-ADDRESS *
```

```
    INSTANCE-NAME
```

```
USER
```

```
  INITIAL-OBJECT
```

```
  UPPIE
```

```
    SUPPIE
```

```
    MUPPIE
```

```
    YUPPIE
```

```
      PUPPIE
```

```
      YUKKIE *
```

```
  CHILD
```

```
    YUKKIE *
```

```
CLIPS>
```

subtrees or subgraphs

```
CLIPS> (browse-classes UPPIE)
```

```
UPPIE
```

```
  SUPPIE
```

```
  MUPPIE
```

```
  YUPPIE
```

```
    PUPPIE
```

```
    YUKKIE *
```

```
CLIPS> (browse-classes YUPPIE)
```

```
YUPPIE
```

```
  PUPPIE
```

```
  YUKKIE *
```

```
CLIPS> (browse-classes YUKKIE)
```

```
YUKKIE *
```

```
CLIPS>
```

Apstraktne klase

- Apstraktne klase služe samo za nasleđivanje i ne mogu imati objekte
- **concrete class** može imati objekte
- Function Meaning
- **ppdefclass** Pretty-print the defclass internal structure
- **undefclass** Eliminate class
- **describe-class** Additional information about classes
- **class-abstractp** Predicate function returns TRUE if the class is abstract

Višestruko nasleđivanje

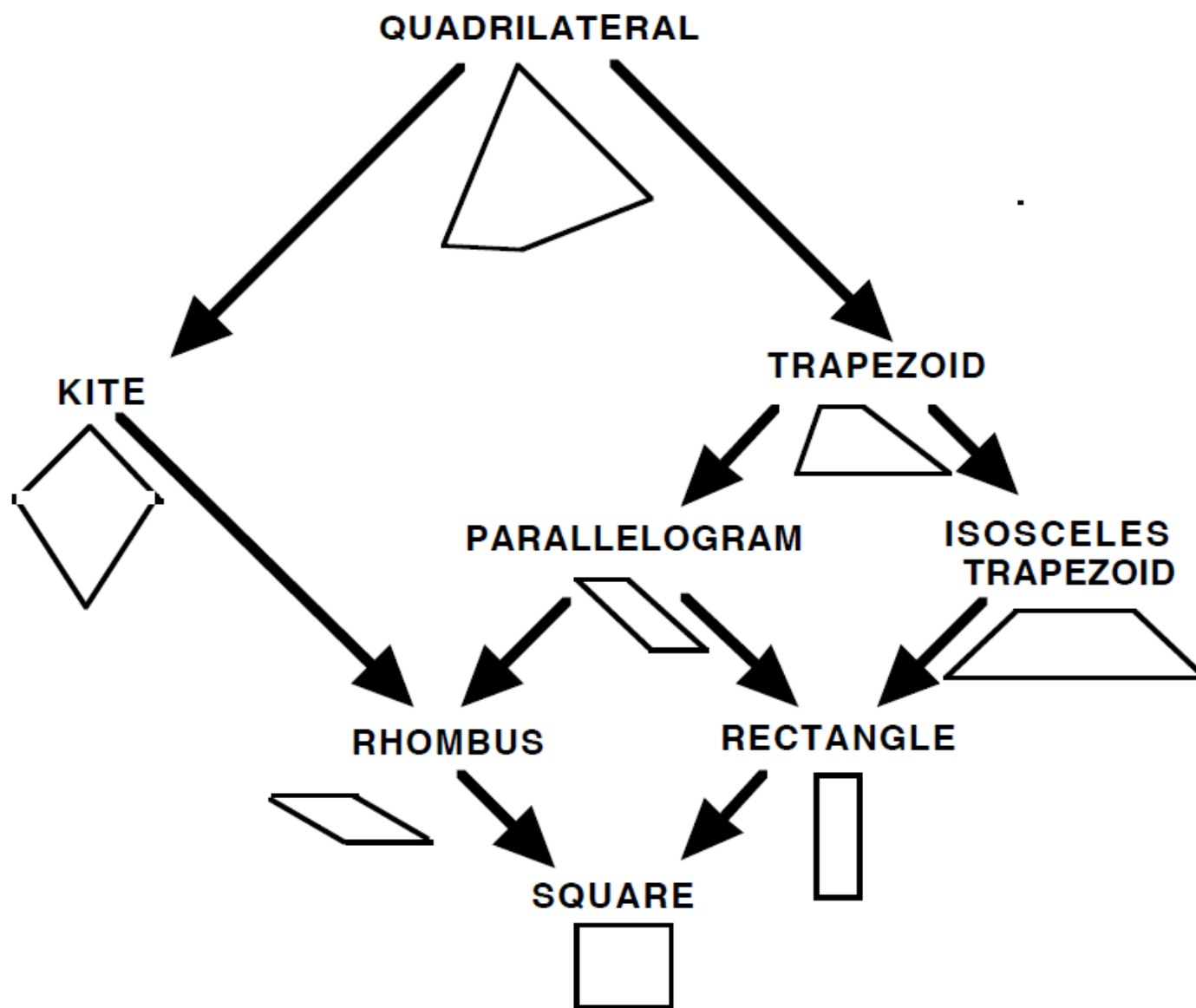


Fig. 1.8 The Quadrilateral Graph

Chapter 9 Meaningful Messages

It's always better to please your superiors than your subordinates.

— Bowen

In this chapter you'll learn more about classes and objects called instances. You will see how to specify the attributes of classes using slots and how to send messages to objects.

Objekti u CLIPS-u

- Objekti se u CLIPS-u označavaju sa [<name>]
 - Između uglastih zagrada se nalazi ime objekta
- | | |
|---------------------------------|------------------|
| Object | Class |
| Dorky_Duck | SYMBOL |
| "Dorky_Duck" | STRING |
| 1.0 | FLOAT |
| 1 | INTEGER |
| (1 1.0 Dorky_Duck "Dorky_Duck") | MULTIFIELD |
| <Pointer: 00AB12CD> | EXTERNAL-ADDRESS |
| [Dorky_Duck] | DUCK |

Objekti i klase

- Klase **SYMBOL**, **STRING**, **FLOAT**, **INTEGER**, i **MULTIFIELD** su tzv. primitivne klase a objekti ovih klasa su primerci primitivnih tipova
- Dve složene klase su **NUMBER** i **LEXEME**
- **NUMBER** obuhvata **INTEGER** i **FLOAT**
- **LEXEME** obuhvata **SYMBOL** i **STRING**
- **user-defined object types** su korisnički definisani tipovi
- **symbol-to-instance-name** funkcija konvertuje podatak tipa **SYMBOL** u ime objekta
- **instance-name-to-symbol** funkcija konvertuje podatak tipa ime objekta u **SYMBOL**

symbol-to-instance-name

instance-name-to-symbol

- Primer za **symbol-to-instance-name** funkciju
- CLIPS> (clear) ; Get rid of any old classes
- CLIPS> (symbol-to-instance-name Dorky_Duck)
- [Dorky_Duck]
- CLIPS> (symbol-to-instance-name (sym-cat Dorky "_" Duck))
- [Dorky_Duck]
- CLIPS>
- Primer za **instance-name-to-symbol** funkciju
- CLIPS> (instance-name-to-symbol [Dorky_Duck])
- Dorky_Duck
- CLIPS> (str-cat (instance-name-to-symbol [Dorky_Duck]) " is a
- DUCK")
- "Dorky_Duck is a DUCK"
- CLIPS>

describe-class – opis klase

- CLIPS> (defclass DUCK (is-a USER))
- CLIPS> (describe-class DUCK)
- =====
- *****
- Concrete: direct instances of this class can be created.
- Reactive: direct instances of this class can match defrule patterns.
- Direct Superclasses: USER
- Inheritance Precedence: DUCK USER OBJECT
- Direct Subclasses:
- -----
- Recognized message-handlers:
- init primary in class USER
- delete primary in class USER
- create primary in class USER
- print primary in class USER
- direct-modify primary in class USER
- message-modify primary in class USER
- direct-duplicate primary in class USER
- message-duplicate primary in class USER
- *****
- =====
- CLIPS>

make-instance – kreiranje objekata

- (make-instance [<instance-name>] of <class> <slot-override>)
- CLIPS> (make-instance [Dorky] of DUCK)
- [Dorky]
- CLIPS> (make-instance [Elsie] of COW)
- [PRNTUTIL1] Unable to find class COW.
- CLIPS> (make-instance Dorky_Duck of DUCK)
- [Dorky_Duck]
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- [Dorky_Duck] of DUCK
- For a total of 3 instances.
- CLIPS>

- CLIPS> (make-instance Dorky_Duck of DUCK)
- [Dorky_Duck] ; Instance Dorky_Duck is made.
- CLIPS>
- (instances) komanda za ispis objekata
- U modulu može postojati samo jedan objekat datog imena
- Klasa ne može biti redefinisana dok postoje objekti te klase
- CLIPS> (make-instance Dorky_Duck of DUCK)
- [Dorky_Duck]
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- [Dorky_Duck] of DUCK ; Still only one Dorky_Duck
- For a total of 3 instances.
- CLIPS>

definstances

- CLIPS> (reset)
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- For a total of 1 instance.
- CLIPS>
- CLIPS> (definstances DORKY_OBJECTS "The Dorky Cousins"
- (Dorky of DUCK)
- (Dorky_Duck of DUCK))
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- For a total of 1 instance.
- CLIPS> (reset)
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- [Dorky_Duck] of DUCK
- For a total of 3 instances.
- CLIPS>

unmake-instance

- CLIPS> (unmake-instance *) ; Delete all instances
- TRUE
- CLIPS> (instances) ; Check that all are gone
- CLIPS> (reset) ; Create new instances again
- CLIPS> (instances) ; Check new instances created
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- [Dorky_Duck] of DUCK
- For a total of 3 instances.
- CLIPS> (unmake-instance [Dorky]) ; Delete a specific instance
- TRUE
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky_Duck] of DUCK
- For a total of 2 instances.
- CLIPS>

send funkcija – slanje poruka objektima

- (send [<instance-name>] <message>)
- U send f-ji se može navesti samo jedan objekat
- CLIPS> (reset) ; Create new instances again
- CLIPS> (instances) ; Check new instances created
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- [Dorky_Duck] of DUCK
- For a total of 3 instances.
- CLIPS> (send [Dorky_Duck] delete)
- TRUE
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky] of DUCK
- For a total of 2 instances.
- CLIPS>

print i put handler-i

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (slot sound)
- (slot age))
- CLIPS> (definstances DORK_OBJECTS
- (Dorky_Duck of DUCK))
- CLIPS> (reset)
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound nil)
- (age nil)
- CLIPS> (send [Dorky_Duck] put-sound quack)
- quack
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (age nil)
- CLIPS>

slot-override in a make-instance

- CLIPS> (make-instance Dixie_Duck of DUCK (sound quack) (age 2))
- [Dixie_Duck]
- CLIPS> (send [Dixie_Duck] print)
- [Dixie_Duck] of DUCK
- (sound quack)
- (age 2)
- CLIPS>
- CLIPS> (send [Dorky_Duck] put-color white) ; No slot color
- [MSGFUN1] No applicable primary message-handlers found for putcolor.
- FALSE
- CLIPS> (send [Dorky_Duck] get-age)
- nil
- CLIPS> (send [Dorky_Duck] put-age 1) ; Value put in age
- 1
- CLIPS> (send [Dorky_Duck] get-age) ; Check value is correct
- 1
- CLIPS>

Class Etiquette - OOP paradigm is class oriented

Rules of Class Etiquette

1. The class hierarchy should be in specialized logical increments using is-a links.
2. A class is unnecessary if it has only one instance.
3. A class should not be named for an instance and *vice versa*.

- 1. ukoliko je samo jedna klasa dovoljna u aplikaciji – programu, onda verovatno klasa uopšte nije ni potrebna, tj. OOP pristup rešavanju problema nije adekvatan
- Klasa nije potrebna ako će postojati samo jedan primerak objekta
- Između klase i objekta treba praviti razliku i u pogledu imena

Predikatske funkcije za slotove

- Function Meaning
- **class-slot-existp** Returns TRUE if the class slot exists
- **slot-existp** Returns TRUE if the instance slot exists
- **slot-boundp** Returns TRUE if the specified slot has a value
- **instance-address** Returns the machine address at which the specified instance is stored.
- **instance-name** Returns the name given an address and *vice versa*
- **instancep** Returns TRUE if its argument is an instance
- **instance-addressp** Returns TRUE if its argument is an instance address
- **instance-namep** Returns TRUE if its argument is an instance name
- **instance-existp** Returns TRUE if instance exists
- **list-definstances** Lists all the definstances
- **ppdefinstances** Pretty-prints the definstance
- **watch instances** Allows you to watch instances being created and deleted.
- **unwatch instances** Turns off watching instances.
- **save-instances** Save instances to a file
- **load-instances** Load instances to a file
- **undefinstances** Deletes the named definstance.

Chapter 10 Fascinating Facets

If you want to have class, then act, dress, and talk like your friends.

In this chapter you'll learn more about slots and how to specify their characteristics by using **facets**. Just as slots describe instances, facets describe slots. The use of facets is good software engineering because there is a greater chance of CLIPS flagging an illegal value rather than risking a runtime error or crash. There are many types of facets that may be used to specify slots, as summarized in the following table.

<i>Facet Name</i>	<i>Description</i>
default and default-dynamic	Set initial values for slots
cardinality	Number of multifold values
access type	Read-write, read-only, initialize-only access
storage	Local slot in instance or shared slot in class
propagation	Inherit, or no inherit slots
source	composite or exclusive inheritance
documentation	Documentation of slots
override-message	Indicate message to send for slot override
create-accessor	Create put- and get- handlers
visibility	Public, or private to defining class only
reactive	Changes to a slot trigger pattern-matching

default facet postavlja default vrednost za slot

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (slot sound (default quack))
- (slot ID)
- (slot sex (default male)))
- CLIPS> (make-instance Dorky_Duck of DUCK)
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (ID nil)
- (sex male)
- CLIPS>

(gensym*) function

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (slot sound (default quack))
- (slot ID (default (gensym*))))
- (slot sex (default male)))
- CLIPS> (make-instance [Dorky_Duck] of DUCK) ; Dorky_Duck #1
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (ID gen1)
- (sex male)
- CLIPS> (make-instance [Dorky_Duck] of DUCK) ; Dorky_Duck #2
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (ID gen1)
- (sex male)
- CLIPS>

default dynamic facet

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (slot sound (default quack))
- (slot ID (default-dynamic (gensym*))))
- (slot sex (default male)))
- CLIPS> (make-instance [Dorky_Duck] of DUCK) ; Dorky_Duck #1
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (ID gen2)
- (sex male)
- CLIPS> (make-instance [Dorky_Duck] of DUCK) ; Dorky_Duck #2
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack)
- (ID gen3) ; Note ID is different from Dorky_Duck #1
- (sex male)
- CLIPS>

Multislot slot

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (multislot sound (default quack quack)))
- CLIPS> (make-instance [Dorky_Duck] of DUCK)
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] print)
- [Dorky_Duck] of DUCK
- (sound quack quack)
- CLIPS>
- CLIPS> (send [Dorky_Duck] put-sound quack1 quack2 quack3)
- (quack1 quack2 quack3)
- CLIPS> (send [Dorky_Duck] get-sound)
- (quack1 quack2 quack3)
- CLIPS>
- CLIPS> (nth\$ 1 (send [Dorky_Duck] get-sound))
- quack1
- CLIPS> (nth\$ 2 (send [Dorky_Duck] get-sound))
- quack2
- CLIPS> (nth\$ 3 (send [Dorky_Duck] get-sound))
- quack3
- CLIPS>

All facets

- `<facet> ::= <default-facet> | <storage-facet> |`
- `<access-facet> | <propagation-facet> |`
- `<source-facet> | <pattern-match-facet> |`
- `<visibility-facet> | <create-accessor-facet>`
- `<override-message-facet> | <constraint-`
`attributes>`
- `<default-facet> ::=`
- `(default ?DERIVE | ?NONE | <expression>*) |`
- `(default-dynamic <expression>*)`

- `<storage-facet> ::= (storage local | shared)`
- `<access-facet>`
 - `::= (access read-write | read-only | initialize-only)`
- `<propagation-facet> ::= (propagation inherit | no-inherit)`
- `<source-facet> ::= (source exclusive | composite)`
- `<pattern-match-facet>`
 - `::= (pattern-match reactive | non-reactive)`
- `<visibility-facet> ::= (visibility private | public)`
- `<create-accessor-facet>`
 - `::= (create-accessor ?NONE | read | write | read-write)`
- `<override-message-facet>`
 - `::= (override-message ?DEFAULT | <message-name>)`
- `<handler-documentation>`
 - `::= (message-handler <name> [<handler-type>])`
- `<handler-type> ::= primary | around | before | after`

Chapter 11 Handling Handlers

There are two steps in learning how to use a shovel: (1) finding which part is the handle, and (2) what to do with the handle.

Handlers are essential in OOP because they support object encapsulation. The only proper way that objects can respond to messages is by having an appropriate handler to receive the message and take appropriate action. In this chapter you'll learn the how messages are interpreted by objects. You'll see how to modify existing message-handlers, and how to write your own.

Message handlers

- CLIPS> (clear)
- CLIPS> (send 1 + 2)
- [MSGFUN1] No applicable primary message-handlers found for +.
- FALSE
- CLIPS>
- CLIPS> (describe-class INTEGER)
- =====
- *****
- Abstract: direct instances of this class cannot be created.
- Direct Superclasses: NUMBER
- Inheritance Precedence: INTEGER NUMBER PRIMITIVE OBJECT
- Direct Subclasses:
- *****
- =====
- CLIPS>

Defmessage handler

- CLIPS>
- (defmessage-handler NUMBER + (?arg) ; Argument of handler
- (+ ?self ?arg)) ; Function addition of handler
- CLIPS> (send 1 + 2)
- 3
- CLIPS> (send 2.5 + 3)
- 5.5
- CLIPS> (send 2.5 + 2.6)
- 5.1
- CLIPS> (describe-class NUMBER)
- =====
- *****
- Abstract: direct instances of this class cannot be created.
- Direct Superclasses: PRIMITIVE
- Inheritance Precedence: NUMBER PRIMITIVE OBJECT
- Direct Subclasses: INTEGER FLOAT
- -----
- Recognized message-handlers:
- + primary in class NUMBER
- *****
- =====
- CLIPS>

Concatenate strings

- CLIPS>
- (defmessage-handler LEXEME + (?arg) ; Argument of handler
- (sym-cat ?self ?arg)); Function concatenation of handler
- CLIPS> (send Dorky_ + "Duck") ; SYMBOL + STRING
- Dorky_Duck ; SYMBOL result
- CLIPS>

USER class handlers

- CLIPS> (clear)
- CLIPS>
- (defclass DUCKLING (is-a USER)
- (slot sound (default quack))
- (slot age (visibility public)))
- CLIPS>
- (defclass DUCK (is-a DUCKLING)
- (slot sound (default quack)))
- CLIPS>
- (definstances DUCKY_OBJECTS
- (Dorky_Duck of DUCK (age 2))
- (Dinky_Duck of DUCKLING (age .1)))
- CLIPS> (reset)
- CLIPS>
- CLIPS> (defmessage-handler USER print-slots ()
- (ppinstance))
- CLIPS> (send [Dorky_Duck] print-slots)
- [Dorky_Duck] of DUCK
- (age 2)
- (sound quack)
- CLIPS>

Handler type before

- CLIPS> (defmessage-handler USER print before ())
- (printout t "*** Starting to print ***" crlf))
- CLIPS> (send [Dorky_Duck] print)
- *** Starting to print ***
- [Dorky_Duck] of DUCK
- (age 2)
- (sound quack)
- CLIPS>

Handler type after

- CLIPS> (defmessage-handler USER print after ())
- (printout t "*** Finished printing ***" crlf))
- CLIPS> (send [Dorky_Duck] print)
- *** Starting to print ***
- [Dorky_Duck] of DUCK
- (age 2)
- (sound quack)
- *** Finished printing ***
- CLIPS>

Defmessage handler

- (defmessage-handler <class-name> <message-name> [handler-type])
- [comment]
- (<parameters>* [wildcard-parameter])
- <action>*)
- handler-type: before, primary, after, around

<i>Handler Type</i>	<i>Class Role</i>	<i>Return Value</i>
around	Set up environment for other handlers	Yes
before	Auxilliary work before primary	No
primary	Perform major task of message	Yes
after	Auxilliary work after primary	No

Summary of Message-Handler Types

Predefined primary handlers of USER

<i>Primary Type</i>	<i>Class Role</i>
init	Initialize an instance
delete	Delete an instance
print	Print the object
direct-modify	Directly modifies slots
message-modify	Modifies slots using put- messages
direct-duplicate	Duplicates instance without put- message
message-duplicate	Duplicates an instance using messages

Predefined USER Message-handlers of Primary Type

- Perdefinisani handler-i se ne mogu menjati, ali se mogu dodati handler-i drugih tipova – befor, after i arround.
- Putem nasleđivanja ovi handler-i su dostupni svim klasama koje nasleđuju USER

modify-instance function uses the direct-modify message

- (defclass A (is-a USER)
- (role concrete)
- (slot foo)
- (slot bar))
- (make-instance a of A)
- (modify-instance a (foo 0)) - delayed pattern matching
- (active-modify-instance a (foo 1)) – immediate pattern matching

message-modify-instance function uses the message-modify message

- (message-modify-instance a (bar 4)) – delayed pattern matching
- (active-message-modify-instance <instance> <slot-override>*) – immediate pattern matching

duplicate-instance function uses the direct-duplicate message

- (duplicate-instance a) – delayed pattern matching
- (active-duplicate-instance <instance> [to <instance-name>] <slot-override>*) – immediate pattern matching

message-duplicate-instance function uses the message-duplicate

- (message-duplicate-instance a to b (bar 6)) –
delayed pattern matching
- (active-message-duplicate-instance <instance>
- [to <instance-name>] <slot-override>*) –
immediate pattern matching

before and after handler types for the **init** **primary handler**

- CLIPS> (defmessage-handler USER init before ())
- (printout t "*** Starting to make instance ***" crlf))
- CLIPS> (defmessage-handler USER init after ())
- (printout t "*** Finished making instance ***" crlf))
- CLIPS> (reset)
- *** Starting to make instance ***
- *** Finished making instance ***
- *** Starting to make instance ***
- *** Finished making instance ***
- *** Starting to make instance ***
- *** Finished making instance ***
- CLIPS> (make-instance Dixie_Duck of DUCK (age 1))
- *** Starting to make instance ***
- *** Finished making instance ***
- [Dixie_Duck]
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Dorky_Duck] of DUCK
- [Dinky_Duck] of DUCKLING
- [Dixie_Duck] of DUCK
- For a total of 4 instances.
- CLIPS>

Self parameter

- ?self:<slot-name> - čitanje slot-a tekućeg aktivnog objekta ili dodeljivanje vrednosti sa f-jom bind
- CLIPS> (defmessage-handler DUCK print-age primary ())
- (printout t "*** Starting to print ***" crlf
- "Age = " ?self:age crlf
- "*** Finished printing ***" crlf))
- CLIPS> (send [Dorky_Duck] print-age)
- *** Starting to print ***
- Age = 2
- *** Finished printing ***
- CLIPS>
- Alternativno, mogu se koristiti dynamic-get i dynamic-put:
- (send ?self dynamic-get-<slot>)
- (send ?self dynamic-put-<slot>)

dynamic-get-

- CLIPS> (defmessage-handler USER print-age primary ())
- (printout t "*** Starting to print ***" crlf
- "Age = " (dynamic-get age) crlf
- "*** Finished printing ***" crlf))
- CLIPS> (send [Dorky_Duck] print-age)
- *** Starting to print ***
- Age = 2
- *** Finished printing ***
- CLIPS>

?self:<slot-name>

- ?self:<slot-name> se određuje statički preko nasleđivanja
- Ukoliko podklasa redefiniše slot, onda message handler nadklase koji za pristup koristi ?self:<slot-name> neće uspeti da pristupi tom slotu.
- Ako se želi dinamički pristup, onda se koriste funkcije dynamic-get- i dynamic-put- koje ako se koriste iz nadklasa i podklasa
- Da bi dinamičke funkcije mogle da pristupe slotu, neophodno je da vidljivost slotu bude javna

Dynamic put

- CLIPS> (clear)
- CLIPS> (defclass DUCK (is-a USER)
- (slot sound (default quack))
- (slot age))
- CLIPS> (defmessage-handler DUCK lie-about-age (?change)
- (bind ?new-age (- ?self:age ?change))
- (dynamic-put age ?new-age)
- (printout t "*** Starting to print ***" crlf
- "I am only " ?new-age crlf
- "*** Finished printing ***" crlf))
- CLIPS> (make-instance [Dorky_Duck] of DUCK (age 3))
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] lie-about-age 2)
- *** Starting to print ***
- I am only 1
- *** Finished printing ***
- CLIPS>

daemon

- Daemon je procedura koja se implicitno izvršava kada se nešto dešava, a ne pozivaju se eksplicitno
- Before i after handler-i su primeri za daemon procedure, jer se izvršavaju kada se pozove primary message handler
- Ovakav način izvršavanja se naziva **declarative implementation** jer se tačno zna kada se izvršavaju before i after handler-i
- **imperative implementation** je drugi način implementacije gde se eksplicitno programira kada će se akcije izvršavati

Around handler

- Around handler je pogodan za **imperative implementation**
- Osnovne faze u izvršavanju around handler-a su:
 - 1. Otpočinje pre bilo kog drugog handler-a
 - 2. Poziva sledeći handler bilo sa **call-next-handler** pri čemu se predaju isti parametri ili sa **override-next-handler** za poziv sledećeg handler-a sa različitim parametrima
 - 3. Nastavlja izvršavanje kada se zadnji handler završi
 - 4. Pošto se bilo koji drugi around, before, primary, ili after handlers završi, around handler nastavlja izvršavanje

Around handler

- keyword *call-next-handler* se koristi za poziv sledećeg handler-a
- Za handler se kaže da je zaklonjen – shadowed, sa **shadower**-om ako se mora pozvati iz shadower-a sa metodom kao što je *call-next-handler*
- Predicate function pod nazivom **next-handlerp** se koristi za proveru da li postoji sledeći handler koji se može pozvati
- CLIPS> (defmessage-handler DUCK lie-about-age around (?change)
- (bind ?old-age ?self:age)
- (if (next-handlerp) then
- (call-next-handler))
- (bind ?new-age ?self:age)
- (if (<> ?old-age ?new-age) then
- (printout t "Dorky_Duck is lying!" crlf
- "Dorky_Duck is lying!" crlf
- "He's really " ?old-age crlf)))

Around handler

- CLIPS> (make-instance [Dorky_Duck] of DUCK (age 3))
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] lie-about-age 1)
- *** Starting to print ***
- I am only 2
- *** Finished printing ***
- Dorky_Duck is lying!
- Dorky_Duck is lying!
- He's really 3
- CLIPS>

override-next-handler function

- CLIPS> (defmessage-handler DUCK lie-about-age around (?change)
- (bind ?old-age ?self:age)
- (if (next-handlerp) then
- (override-next-handler (/ ?change 2))) ; Divide age; in half!
- (bind ?new-age ?self:age)
- (if (<> ?old-age ?new-age) then
- (printout t "Dorky_Duck is lying!" crlf
- "Dorky_Duck is lying!" crlf
- "He's really " ?old-age crlf)))
- CLIPS> (make-instance [Dorky_Duck] of DUCK (age 3))
- [Dorky_Duck]
- CLIPS> (send [Dorky_Duck] lie-about-age 1)
- *** Starting to print ***
- I am only 2.5
- *** Finished printing ***
- Dorky_Duck is lying!
- Dorky_Duck is lying!
- He's really 3
- CLIPS>

Around handler

- Povratna vrednost od call-next-handler i override-next-handler je vrednost koju vraća shadowed handler
- Uvek mora da postoji primary handler uz koji mogu postojati i ostali handler-i
- Pravila za predaju poruka – poziv handler-a:

Rules of Message-handler Precedence

1. All the around handlers start execution. Then the before, primary, and after handlers start and finish, followed by completion of the arounds execution.
2. The around, before, and primary handlers are called in order of highest precedence class to lowest.
3. The after handlers are called from lowest precedence class to highest.
4. Each around handler must explicitly call the next shadowed handler.
5. Higher precedence primaries must explicitly call lower precedence (shadowed) primaries if they are to execute.

Around handler

- Pošto samo around i primary handler-i mogu da vraćaju vrednosti, i around shadows – zaklanjaju primarne, sledi da je povratna vrednost od (send) u stvari povratna vrednost od around
- Ako ne postoji around, onda će povratna vrednost biti od primarnog handler-a najvišeg prioriteta
- Sledeća tabela sumira povratne vrednosti raznih tipova handler-a

<i>Handler Type</i>	<i>Return Value</i>
around	Ignore or capture return value of next most specific around or primary
before	Ignore. Side-effect only
primary	Ignore or capture return value of more general primary
after	Ignore. Side-effect only.

Još o nasleđivanju

- Nasleđivanje je značajan i moćan mehanizam OOP kojim se formiraju novi složeniji pojmovi – klase oslanjanjem i preuzimanjem članova već gotovih i istestiranih klasa
- Najčešći vid nasleđivanja je nasleđivanje tipa **is a** ili **is a kind of** čime se formiraju novi složeniji pojmovi iste vrste
- Novi pojmovi su na nižem nivou apstrakcije, ali sa više detalja – članova klase
- Najsloženije klase u pogledu detalja su upravo klase na najnižem nivou hijerarhije - apstrakcije

Još o nasleđivanju

- Nasleđivanje tipa **is a** ili **kind of** je nasleđivanje kojim se uspostavljaju veze između klasa (pojmova - tipova entiteta) u smeru od opštije klase – roditeljske prema klasi koja nasleđuje i koja je manje opšteg tipa u odnosu na klasu od koje nasleđuje – **Inheritance by Specialization**
- Moguća je i drugačija vrsta nasleđivanja – drugog tipa pri čemu je naziv za taj tip nasleđivanja **is made of** – sastoji se od, i vrši se od klase – pojma koji je jednostavniji ka pojmu koji je složeniji i koji se logično izvodi iz jednostavnijeg pojma – **Inheritance by Generalization**

Inheritance by Specialization

- Primer za uobičajeno nasleđivanje po opadajućoj generalizaciji – apstrakciji i rastućoj složenosti
- Tačka – point, Line – linija, Triangle - trougao
- (defclass POINT (is-a USER)
- (multislot point1))
- (defclass LINE (is-a POINT)
- (multislot point2))
- (defclass TRIANGLE (is-a LINE)
- (multislot point3))

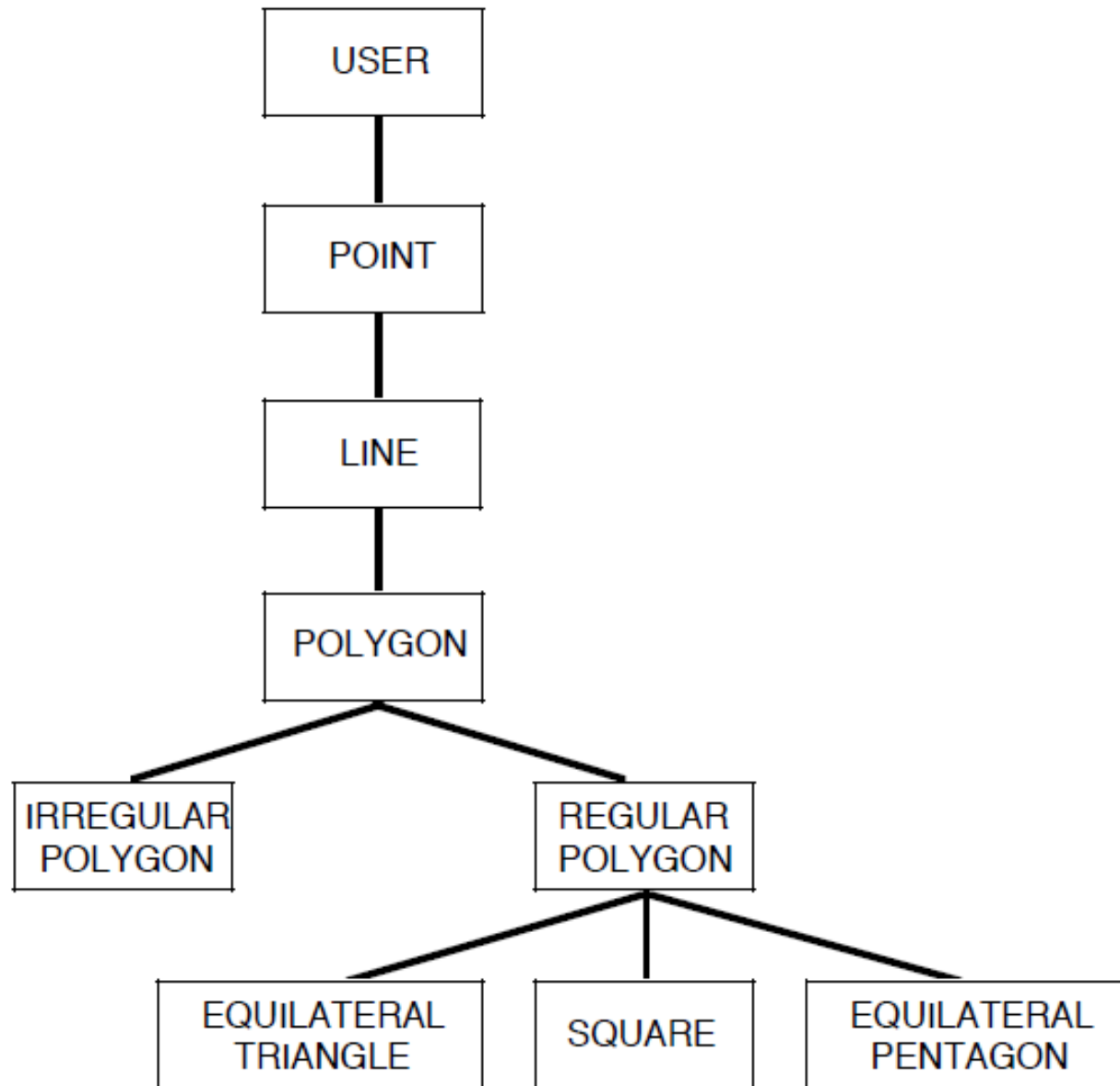
Još o nasleđivanju 2

- Klasično nasleđivanje je pogodno za operacije analize kada se vrši raščlanjavanje pojmova na višem nivou apstrakcije formiranjem pojmova na nižem nivou apstrakcije kada se formiraju pojmovi – klase sa više detalja nego pojmovi na višem hijerarhijskom nivou.
- Nasleđivanje kod koga se kreiraju složenije klase – pojmovi od jednostavnijih je pogodno za predstavljanje sinteze kojom se formiraju složeni pojmovi
- Ali, ti složeni pojmovi NISU na višem nivou apstrakcije !!!

Još o nasleđivanju

- Uobičajeno nasleđivanje - **by Specialization**, je uvek usmereno ka hijerarhijski nižim klasama.
- Druga vrsta nasleđivanja **by Generalization**, u stvari nije vezana za apstrakciju - hijerarhiju, već za odnos sastoji se od
- Implementaciona razlika između ove dve vrste nasleđivanja je u tome, da naredna klasa ima nove slotove koji nisu tipa slotova koji postoje kod klase roditelja, već ima slotove koji su tipa klase od koje se nasleđuje

Primer za odnos nasleđivanja is made of



Implementacija odnosa nasleđivanja is made of

Definicija osnovne klase POINT

- CLIPS> (clear)
- CLIPS> (defclass POINT (is-a USER)
- (multislot position (propagation no-inherit)))
- CLIPS>
- Klasa POINT nasleđuje od predefinisane USER klase
- Facet **propagation** ima vrednost **no-inherit** čiji je smisao u tome da spreči da naredna klasa nasledi multislot position – implementaciona razlika u odnosu na uobičajeni odnos nasleđivanja

Definicija klase LINE koja nasleđuje od POINT

- (defclass LINE (is-a POINT)
- (slot point1 (default-dynamic (make-instance (gensym*) of
- POINT))
- (propagation no-inherit))
- (slot point2 (default-dynamic (make-instance (gensym*) of
- POINT))
- (propagation no-inherit))
- (message-handler find-point)
- (message-handler print-points)
- (message-handler find-distance)
- (message-handler print-distance))

Kreiranje objekata

- CLIPS> (definstances LINE_OBJECTS
- (Line1 of LINE))
- CLIPS> (reset)
- CLIPS>
- (send (send [Line1] get-point1) put-position 0)
- (0)
- CLIPS>
- (send (send [Line1] get-point2) put-position 1)
- (1)
- CLIPS> (send [Line1] print)
- [Line1] of LINE
- (point1 [gen1])
- (point2 [gen2])
- CLIPS>
- (send (send [Line1] get-point1) get-position)
- (0)
- CLIPS>
- (send (send [Line1] get-point2) get-position)
- (1)
- CLIPS>

Kreiranje procedura članova, tj. message handler-a

- CLIPS> (defmessage-handler LINE find-point (?point)
- (send (send ?self (sym-cat "get-point" ?point)) get-position))
- CLIPS> (defmessage-handler LINE print-points ()
- (printout t "point1 " (send ?self find-point 1) crlf
- "point2 " (send ?self find-point 2) crlf))
- CLIPS> (send [Line1] find-point 1)
- (0)
- CLIPS> (send [Line1] find-point 2)
- (1)
- CLIPS>

Testiranje dvo dimenzionalnog slučaja

- Multislot definicija omogućava direktnu generalizaciju bez ikakvih izmena na N dimenzija
- CLIPS> (send (send [Line1] get-point1) put-position 1 2)
- (1 2)
- CLIPS> (send (send [Line1] get-point2) put-position 4 6)
- (4 6)
- CLIPS> (send [Line1] print)
- [Line1] of LINE
- (point1 [gen1])
- (point2 [gen2])
- CLIPS> (send (send [Line1] get-point1) get-position)
- (1 2)
- CLIPS> (send (send [Line1] get-point2) get-position)
- (4 6)
- CLIPS>

- CLIPS> (defmessage-handler LINE find-distance ())
- (bind ?sum 0)
- (bind ?index 1)
- (bind ?len (length (send ?self find-point 1)))
- (bind ?Point1 (send ?self find-point 1))
- (bind ?Point2 (send ?self find-point 2))
- (while (<= ?index ?len)
- (bind ?dif (- (nth ?index ?Point1)
- (nth ?index ?Point2))))
- (bind ?sum (+ ?sum (* ?dif ?dif))))
- (bind ?index (+ ?index 1)))
- (bind ?distance (sqrt ?sum)))
- CLIPS> (defmessage-handler LINE print-distance ())
- (printout t "Distance = " (send ?self find-distance) crlf))
- CLIPS> (send [Line1] print-distance)
- Distance = 5.0
- CLIPS>

Nastavak, definisanje klase trougao

- CLIPS>
- (defclass TRIANGLE (is-a LINE)
- (slot line1 (default-dynamic (make-instance (gensym*) of
- LINE))
- (propagation no-inherit))
- (slot line2 (default-dynamic (make-instance (gensym*) of
- LINE))
- (propagation no-inherit))
- (slot line3 (default-dynamic (make-instance (gensym*) of
- LINE))
- (propagation no-inherit)))
- CLIPS>

Hijerarhija objekata trougla

- CLIPS> (definstances TRIANGLE_OBJECTS
- (Triangle1 of TRIANGLE))
- CLIPS> (reset)
- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [Triangle1] of TRIANGLE
- [gen3] of LINE
- [gen4] of POINT
- [gen5] of POINT
- [gen6] of LINE
- [gen7] of POINT
- [gen8] of POINT
- [gen9] of LINE
- [gen10] of POINT
- [gen11] of POINT
- For a total of 11 instances.

Malo jasnija hijerarhija objekata

trougla

- CLIPS> (send [Triangle1] print)
- [Triangle1] of TRIANGLE
- (line1 [gen3])
- (line2 [gen6])
- (line3 [gen9])
- CLIPS> (send [gen3] print)
- [gen3] of LINE
- (point1 [gen4])
- (point2 [gen5])
- CLIPS>

Zadavanje x i y koordinata

- CLIPS> (send (send (send [Triangle1] get-line1) get-point1) put-position -1 0)
- (-1 0)
- CLIPS> (send (send (send [Triangle1] get-line1) get-point2) put-position 0 2)
- (0 2)
- CLIPS> (send (send (send [Triangle1] get-line2) get-point1) put-position 0 2)
- (0 2)
- CLIPS> (send (send (send [Triangle1] get-line2) get-point2) put-position 1 0)
- (1 0)
- CLIPS> (send (send (send [Triangle1] get-line3) get-point1) put-position 1 0)
- (1 0)
- CLIPS> (send (send (send [Triangle1] get-line3) get-point2) put-position -1 0)
- (-1 0)
- CLIPS>

Ispis koordinata

- CLIPS> (send [Triangle1] print)
- [Triangle1] of TRIANGLE
- (line1 [gen3])
- (line2 [gen6])
- (line3 [gen9])
- CLIPS> (send [gen3] print)
- [gen3] of LINE
- (point1 [gen4])
- (point2 [gen5])
- CLIPS> (send [gen4] print)
- [gen4] of POINT
- (position -1 0)
- CLIPS> (send [gen5] print)
- [gen5] of POINT
- (position 0 2)
- CLIPS>

Handler-i...

- Očitavanje koordinata za tačku trougla
- (send (send (send [Triangle1] get-line1)
- get-point1) get-position)
- CLIPS>
- Analogno definisanje handler-a za trougao
- CLIPS>
- (defmessage-handler TRIANGLE find-line-point
- (?line ?point)
- (send (send (send ?self (sym-cat "get-line" ?line))
- (sym-cat "get-point" ?point)) get-position))
- CLIPS>

Trogao print handler

- CLIPS>
- (defmessage-handler TRIANGLE print-line (?line)
- (printout t "point1 " (send ?self find-line-point ?line 1) crlf "point2 " (send ?self find-line-point ?line 2)
- crlf))
- CLIPS> (send [Triangle1] print-line 1)
- point1 (-1 0)
- point2 (0 2)
- CLIPS>

find-line za TRIANGLE

- CLIPS>
- (defmessage-handler TRIANGLE find-line (?line)
- (send (send (send ?self (sym-cat "get-line" ?line))
- get-point1) get-position)
- (send (send (send ?self (sym-cat "get-line" ?line))
- get-point2) get-position))
- CLIPS> (send [Triangle1] find-line 1)
- (0 2)
- CLIPS>
- Vraća se samo zadnja vrednost

Modifikovani find-line za TRIANGLE

- CLIPS>
- (defmessage-handler TRIANGLE find-line (?line)
- (create\$ (send (send (send ?self (sym-cat "get-line"
- ?line)) get-point1) get-position)
- (send (send (send ?self (sym-cat "get-line"
- ?line)) get-point2) get-position)))
- CLIPS> (send [Triangle1] find-line 1)
- (-1 0 0 2)
- CLIPS>

Other Features

A number of other functions are useful with handlers. Some of these follow.

<u>Function</u>	<u>Meaning</u>
undefmessage-handler	Deletes a specified handler
list-defmessage-handlers	Lists the handlers
delete-instance	Operates on the active instance
message-handler-existp	Returns TRUE if handler exists, else FALSE

The **grouping functions** group COOL items into a multifold variable.

<u>Function</u>	<u>Meaning</u>
get-defmessage-handler-list	Groups the class names, message names and types (direct or inherited)
class-superclasses	Groups all superclass names (direct or inherited.)
class-subclasses	Groups all subclass names (direct or inherited).
class-slots	Groups all slot names (explicitly defined or inherited)
slot-existp	Returns TRUE if class slot exists, else FALSE
slot-facets	Groups the specified slot facet values of a class
slot-sources	Groups the slot names of classes which contribute to a slot in the specified class

FunctionMeaning**object-pattern-match-delay**

Delay pattern matching of rules until after instances are created, modified, or deleted

modify-instance

Modifies instance using slot overrides. Object pattern matching delayed until after modifications.

active-modify-instance

Change the values of the instance concurrent with object pattern matching with *direct-modify* message

message-modify-instance

Change the values of the instance. Delay object pattern matching until all slots are changed

active-message-modify-instance

Change the values of the instance concurrent with object pattern matching using *message-modify*

Chapter 12 Questions and Answers

The best way to learn is by asking yourself questions; the best way to be sorry is by answering all of them.

In this chapter you'll learn how to pattern match on instances. One way is with rules. Also, CLIPS has a number of query functions to match instances. In addition, control facts and slot daemons can be used for pattern matching.

Pattern matching sa objektima

- Pravila mogu da se aktiviraju sa objektima, analogno kao i sa činjenicama
- CLIPS> (clear)
- CLIPS>
- (defclass DUCK (is-a USER)
- (multislot sound (default quack quack)))
- CLIPS> (make-instance [Dorky_Duck] of DUCK)
- [Dorky_Duck]
- CLIPS> (make-instance [Dinky_Duck] of DUCK)
- [Dinky_Duck]
- CLIPS> (defrule find-sound
- ?duck <- (object (is-a DUCK)(sound \$?find))
- =>
- (printout t "Duck " (instance-name ?duck) " says " ?find
- crlf))
- CLIPS> (run)
- Duck [Dinky_Duck] says (quack quack)
- Duck [Dorky_Duck] says (quack quack)
- CLIPS>

Object pattern matching sa name

- CLIPS> (defrule find-sound
- ?duck <- (object (is-a DUCK)(sound \$?find)(name
- [Dorky_Duck]))
- =>
- (printout t "Duck " (instance-name ?duck) " says "
- ?find
- crlf))
- CLIPS> (run)
- Duck [Dorky_Duck] says (quack quack)
- CLIPS>

Baza podataka objekata

- COOL ima specijalne query funkcije za upite nad objektima trenutno prisutnim u memoriji CLIPS-a – instance set

Function	Purpose
any-instancep	Determines if one or more instance-sets satisfy a query
find-instance	Returns the first instance-set that satisfies a query
find-all-instances	Groups and returns <u>all</u> instance-sets which satisfy a query
do-for-instance	Performs an action for the first instance-set which satisfies a query
do-for-all-instances	Performs an action for every instance-set which satisfies a query as they are found
delayed-do-for-all-instances	Groups all instance-sets which satisfy a query and then iterates an action over this group

Instance-set Query Functions

- Slede definicije klasa koje će se koristiti u primerima za prikaz navedenih QUERY funkcija

Sintaksa query funkcija

Example

Instance-set member class restrictions

```
CLIPS>
(do-for-all-instances
  ((?car1 MASERATI BMW) (?car2 ROLLS-ROYCE))
  (> ?car1:price (* 1.5 ?car2:price)
    (printout t ?car1 crlf))
[Albert-Maserati]
CLIPS>
```

Instance-set template

Instance-set query

Instance-set distributed action

Instance-set member variables

Syntax

```
<instance-set-template>
  ::= (<instance-set-member-template>+)
<instance-set-member-template>
  ::= (<instance-set-member-variable> <class-restrictions>)
<instance-set-member-variable> ::= <single-field-variable>
<class-restrictions>           ::= <class-name-expression>+
```

Syntax

<query> ::= <boolean-expression>

Action Syntax

<action> ::= <expression>

- **Example**
- CLIPS>
- (defclass PERSON (is-a USER)
- (role abstract)
- (slot sex (access read-only)
- (storage shared))
- (slot age (type NUMBER)
- (visibility public)))
- CLIPS>
- (defmessage-handler PERSON put-age (?value)
- (dynamic-put age ?value))

- CLIPS>
- (defclass FEMALE (is-a PERSON)
- (role abstract)
- (slot sex (source composite)
- (default female)))
- CLIPS>
- (defclass MALE (is-a PERSON)
- (role abstract)
- (slot sex (source composite)
- (default male)))
- CLIPS>
- (defclass GIRL (is-a FEMALE)
- (role concrete)
- (slot age (source composite)
- (default 4)
- (range 0.0 17.9)))

- CLIPS>
- (defclass WOMAN (is-a FEMALE)
- (role concrete)
- (slot age (source composite)
- (default 25)
- (range 18.0 100.0)))
- CLIPS>
- (defclass BOY (is-a MALE)
- (role concrete)
- (slot age (source composite)
- (default 4)
- (range 0.0 17.9)))
- CLIPS>
- (defclass MAN (is-a MALE)
- (role concrete)
- (slot age (source composite)
- (default 25)
- (range 18.0 100.0)))
- CLIPS>

- CLIPS>
- (definstances PEOPLE
- (Man-1 of MAN (age 18))
- (Man-2 of MAN (age 60))
- (Woman-1 of WOMAN (age 18))
- (Woman-2 of WOMAN (age 60))
- (Woman-3 of WOMAN)
- (Boy-1 of BOY (age 8))
- (Boy-2 of BOY)
- (Boy-3 of BOY)
- (Boy-4 of BOY)
- (Girl-1 of GIRL (age 8))
- (Girl-2 of GIRL))
- CLIPS> (reset)
- CLIPS>

any-instancep

- **Syntax**
- (any-instancep <instance-set-template> <query>)
- **Example**
- Are there any men over age 30?
- CLIPS> (any-instancep ((?man MAN)) (> ?man:age 30))
- TRUE
- CLIPS>

find-instance

- **Syntax**
- (find-instance <instance-set-template> <query>)
- **Example**
- Find the first pair of a man and a woman who have the same age.
- CLIPS>
- (find-instance ((?m MAN) (?w WOMAN)) (= ?m:age ?w:age))
- ([Man-1] [Woman-1])

find-all-instances

- **Syntax**
- (find-all-instances <instance-set-template>
<query>)
- **Example**
- Find all pairs of a man and a woman who have the same age.
- CLIPS>
- (find-all-instances ((?m MAN) (?w WOMAN)) (= ?m:age ?w:age))
- ([Man-1] [Woman-1] [Man-2] [Woman-2])
- CLIPS>

do-for-instance

- **Syntax**
- (do-for-instance <instance-set-template> <query> <action>*)
- **Example**
- Print out the first triplet of different people that have the same age. The calls to **neq** in the query eliminate the permutations where two or more members of the instance-set are identical.
- CLIPS>
- (do-for-instance ((?p1 PERSON) (?p2 PERSON) (?p3 PERSON))
- (and (= ?p1:age ?p2:age ?p3:age)
- (neq ?p1 ?p2)
- (neq ?p1 ?p3)
- (neq ?p2 ?p3))
- (printout t ?p1 " " ?p2 " " ?p3 crlf))
- [Girl-2] [Boy-2] [Boy-3]
- CLIPS>

do-for-all-instances

- **Syntax**
- (do-for-all-instances <instance-set-template> <query> <action>*)
- **Example**
- Print out all triplets of different people that have the same age. The calls to **str-compare** limit the instance-sets which satisfy the query to combinations instead of permutations. Without these restrictions, two instance-sets which differed only in the order of their members would both satisfy the query.
- CLIPS>
- (do-for-all-instances ((?p1 PERSON) (?p2 PERSON) (?p3 PERSON))
- (and (= ?p1:age ?p2:age ?p3:age)
- (> (str-compare ?p1 ?p2) 0)
- (> (str-compare ?p2 ?p3) 0))
- (printout t ?p1 " " ?p2 " " ?p3 crlf))
- [Girl-2] [Boy-3] [Boy-2]
- [Girl-2] [Boy-4] [Boy-2]
- [Girl-2] [Boy-4] [Boy-3]
- [Boy-4] [Boy-3] [Boy-2]
- CLIPS>

delayed-do-for-all-instances

- **Syntax**
- (delayed-do-for-all-instances <instance-set-template>
- <query> <action>*)
- Delete all boys with the greatest age. The test in this case is another query function which determines if there are any older boys than the one currently being examined. The action needs to be delayed until all boys have been processed, or the greatest age will decrease as the older boys are deleted.

delayed-do-for-all-instances

- CLIPS> (watch instances)
- CLIPS>
- (delayed-do-for-all-instances ((?b1 BOY))
- (not (any-instancep ((?b2 BOY))
- (> ?b2:age ?b1:age)))
- (send ?b1 delete))
- <== instance [Boy-1] of BOY
- TRUE
- CLIPS> (unwatch instances)
- CLIPS> (reset)
- CLIPS> (watch instances)
- CLIPS>
- (do-for-all-instances ((?b1 BOY))
- (not (any-instancep ((?b2 BOY))
- (> ?b2:age ?b1:age)))
- (send ?b1 delete))
- <== instance [Boy-1] of BOY
- <== instance [Boy-2] of BOY
- <== instance [Boy-3] of BOY
- <== instance [Boy-4] of BOY
- TRUE
- CLIPS> (unwatch instances)
- CLIPS>

- CLIPS>
- (defclass DEVICE (is-a USER)(role abstract)
- (slot type (access initialize-only));Sensor type
- (slot loc (access initialize-only));Location
- CLIPS>
- (defclass SENSOR (is-a DEVICE)(role concrete)
- (slot reading)
- (slot min (access initialize-only)) ;Min reading
- (slot max (access initialize-only)) ;Max reading
- (slot app (access initialize-only));SEN. APP.
- CLIPS>
- (defclass APPLIANCE (is-a DEVICE)(role concrete)
- (slot setting) ; Depends on appliance
- (slot status)) ; off or on
- CLIPS>

- CLIPS>
- (definstances ENVIRONMENT_OBJECTS
- (T1 of SENSOR (type temperature)
- (loc kitchen)
- (reading 110) ; Too hot
- (min 20)
- (max 100)
- (app FR))
- (T2 of SENSOR (type temperature)
- (loc bedroom)
- (reading 10) ; Too cold
- (min 20)
- (max 100)
- (app FR))

- (S1 of SENSOR (type smoke)
- (loc bedroom)
- (reading nil) ; Bad sensor nil reading
- (min 1000)
- (max 5000)
- (app SA))
- (W1 of SENSOR (type water)
- (loc basement)
- (reading 0) ; OK
- (min 0)
- (max 0)
- (app WP))

- (FR of APPLIANCE (type furnace)
- (loc basement)
- (setting low) ; low or high
- (status on))
- (WP of APPLIANCE (type water_pump)
- (loc basement)
- (setting fixed)
- (status off))
- (SA of APPLIANCE (type smoke_alarm)
- (loc basement)
- (setting fixed)
- (status off)))
- CLIPS>

Query funkcije COOL-a

- Query funkcije se koriste za efikasniji rad sa objektima, tj. sa podacima tipa objekata
- Ove funkcije se direktno mogu koristiti za dobijanje odgovora na sledeća pitanja:
- Koji se sve objekti nalaze u bazi podataka?
- Kako su ti objekti raspoređeni?
- Koji su odnosi između objekata?
- Koji sve objekti određenog tipa postoje – DEVICE, APPLIANCE, SENSOR
- Ima li SENSOR-a tipa temperature, sa regularnom vrednošću između min i max, ima li uopšte ijedan objekat tipa SENSOR?

any-instancep je predicate function

- CLIPS> (instances)
- [initial-object] of INITIAL-OBJECT
- [T1] of SENSOR
- [T2] of SENSOR
- [S1] of SENSOR
- [W1] of SENSOR
- [FR] of APPLIANCE
- [WP] of APPLIANCE
- [SA] of APPLIANCE
- For a total of 8 instances.
- CLIPS> (any-instancep ((?ins SENSOR)) TRUE)
- TRUE ;Function returns TRUE because there is a SENSOR instance
- CLIPS> (any-instancep ((?ins DUCK)) TRUE) ; Evaluation error—Bad!
- [PRNTUTIL1] Unable to find class DUCK. ; No DUCK class
- CLIPS>

defgeneric – defmethod

- CLIPS> (> "duck2" "duck1")
- [ARGACCES5] Function > expected argument #1 to be of type integer or float
- CLIPS>
- CLIPS> (defgeneric >) ; Header declaraction. Actually
- unnecessary
- CLIPS> (defmethod > ((?a STRING) (?b STRING))
- (> (str-compare ?a ?b) 0))
- CLIPS> (> "duck2" "duck1") ; The overload ">" works correctly
- TRUE ; in all three cases.
- CLIPS> (> "duck1" "duck1")
- FALSE
- CLIPS> (> "duck1" "duck2")
- FALSE
- CLIPS>