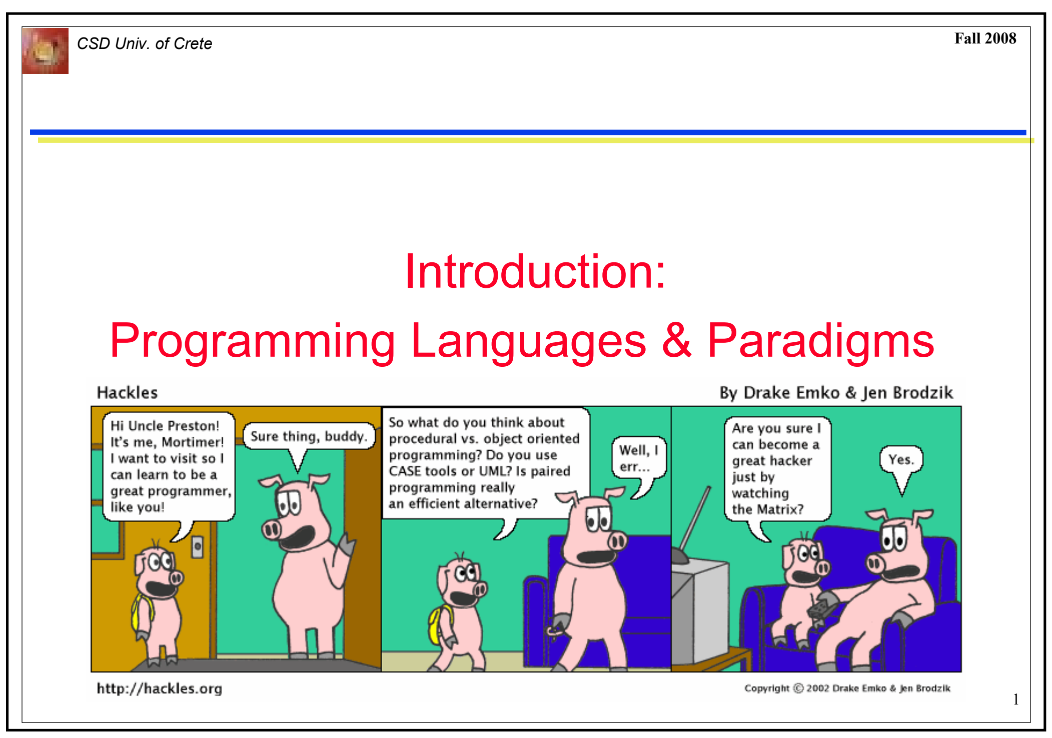
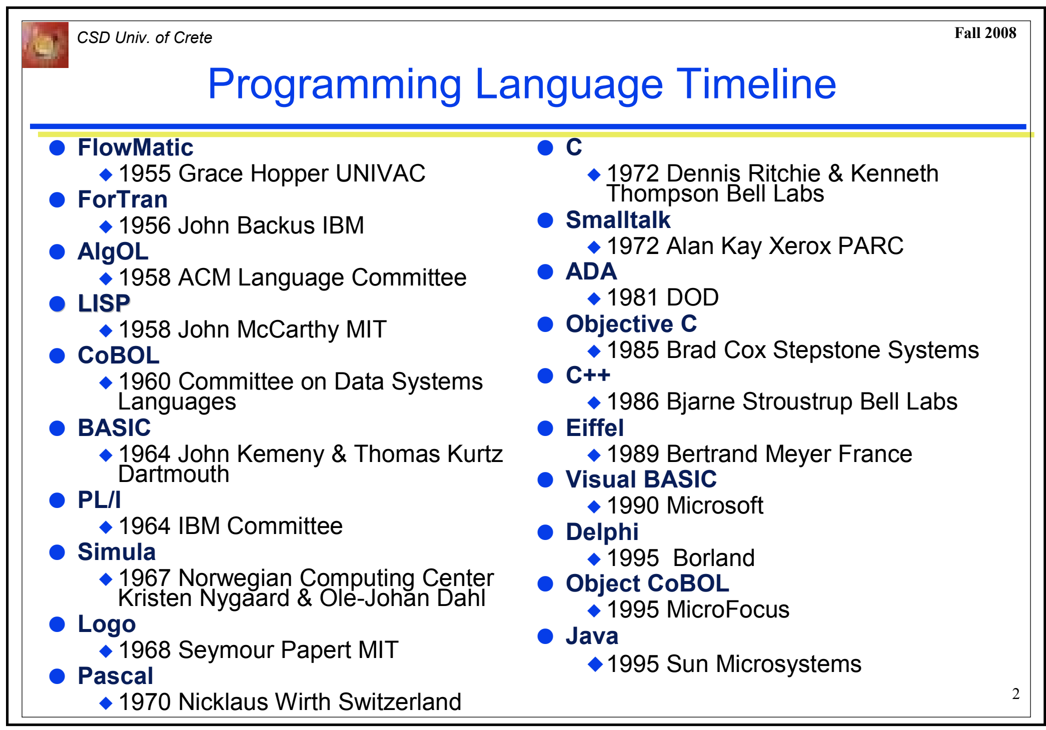
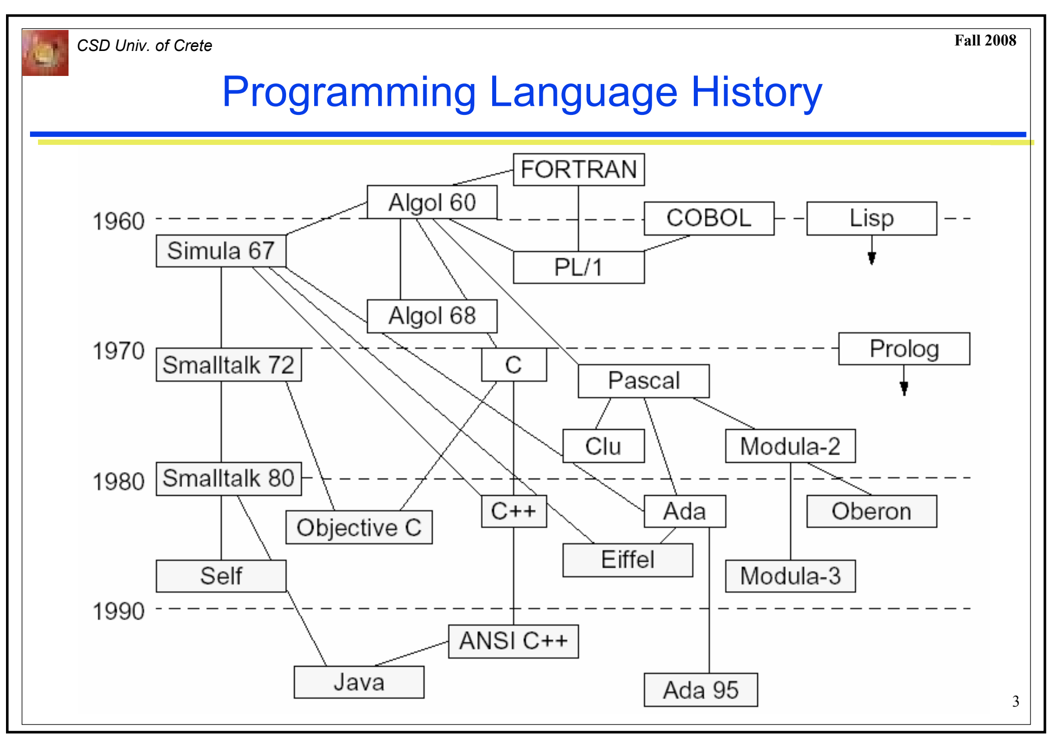
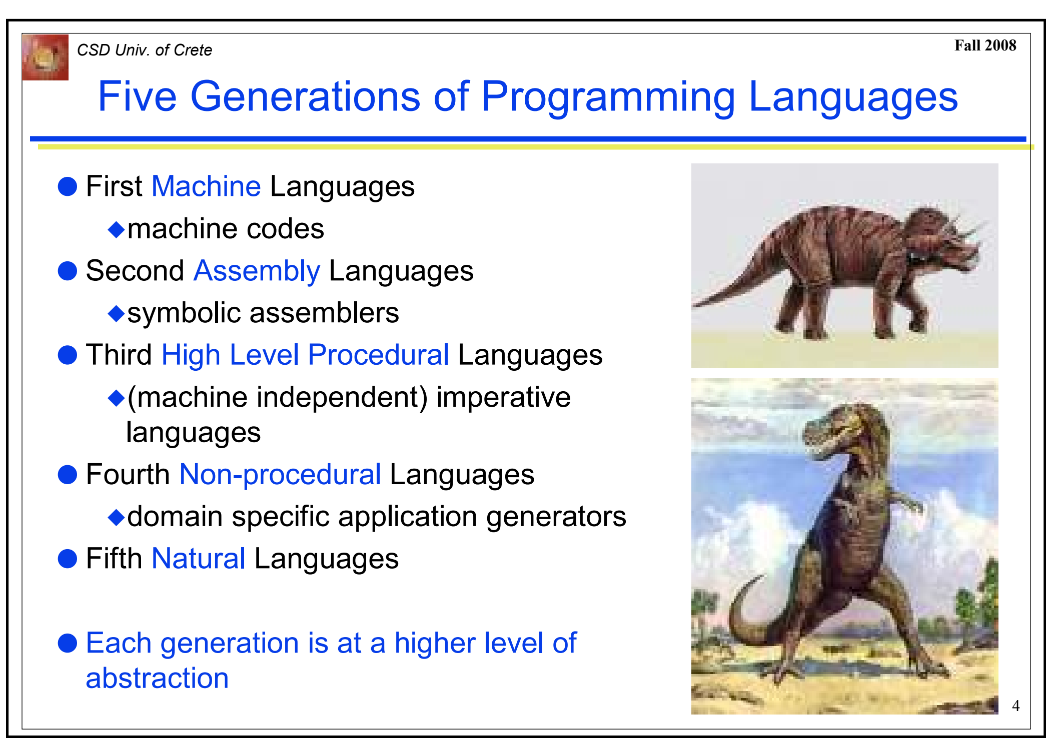
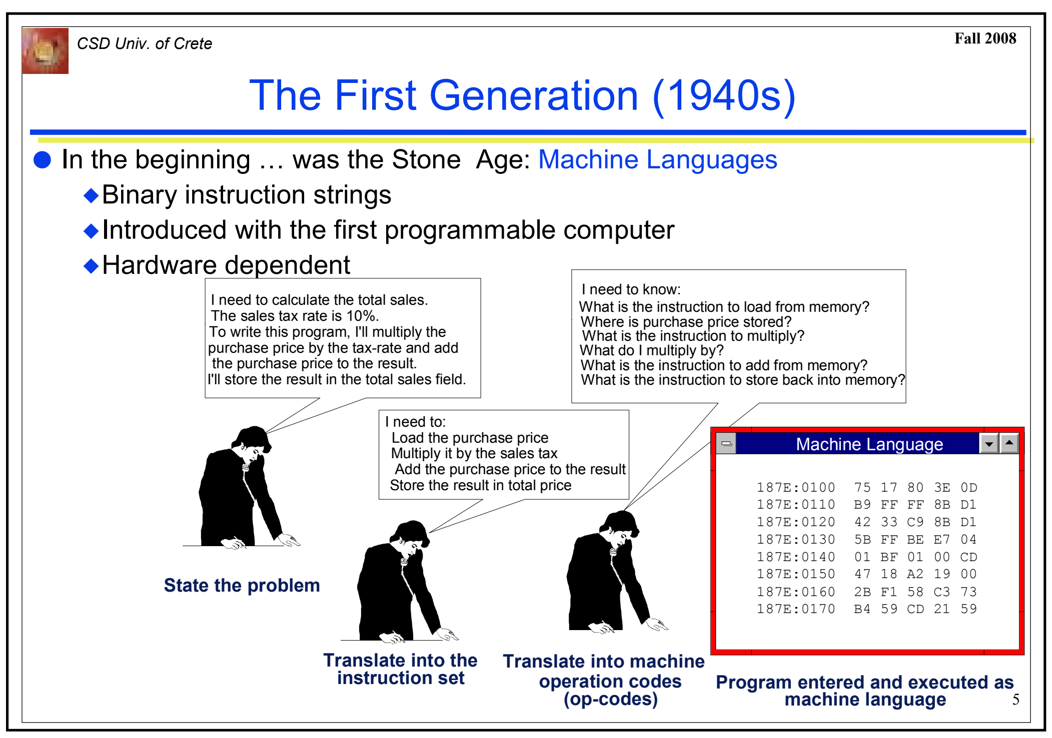
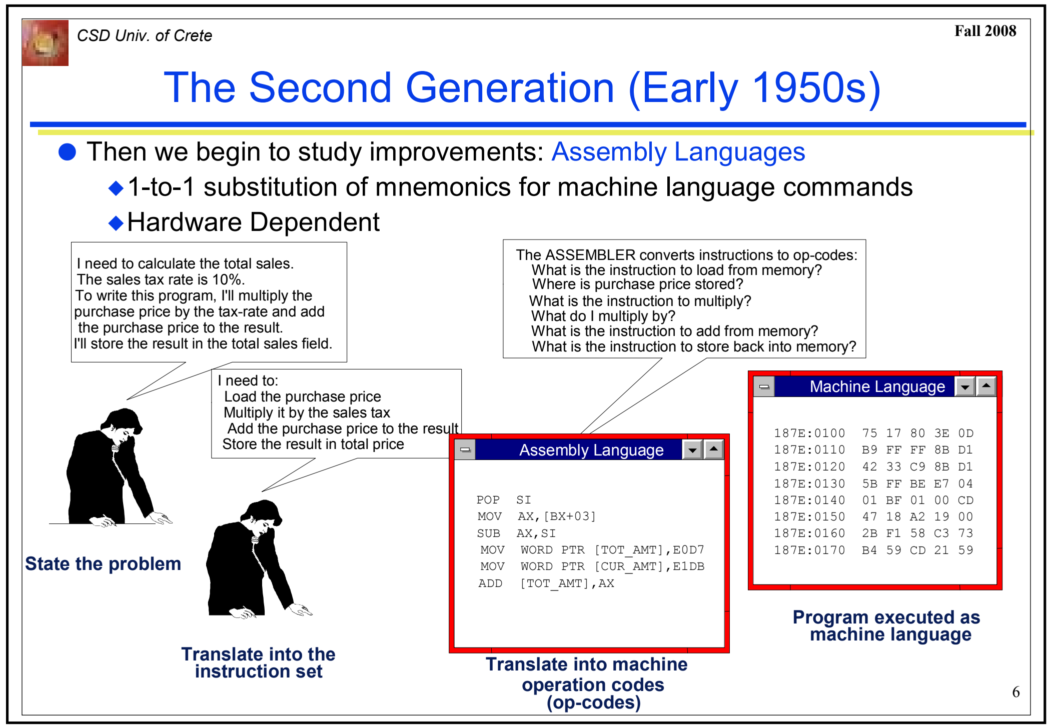

CSD Univ. of Crete
Fall 2008
Introduction:
Programming Languages & Paradigms
Hackles
By Drake Emko & Jen Brodzik
Hi Uncle Preston! It's me, Mortimer! I want to visit so I can learn to be a great programmer, like you!
Sure thing, buddy.
So what do you think about procedural vs. object oriented programming? Do you use CASE tools or UML? Is paired programming really an efficient alternative?
Well, I err...
Are you sure I can become a great hacker just by watching the Matrix?
Yes.
http://hackles.org
Copyright © 2002 Drake Emko & Jen Brodzik
1

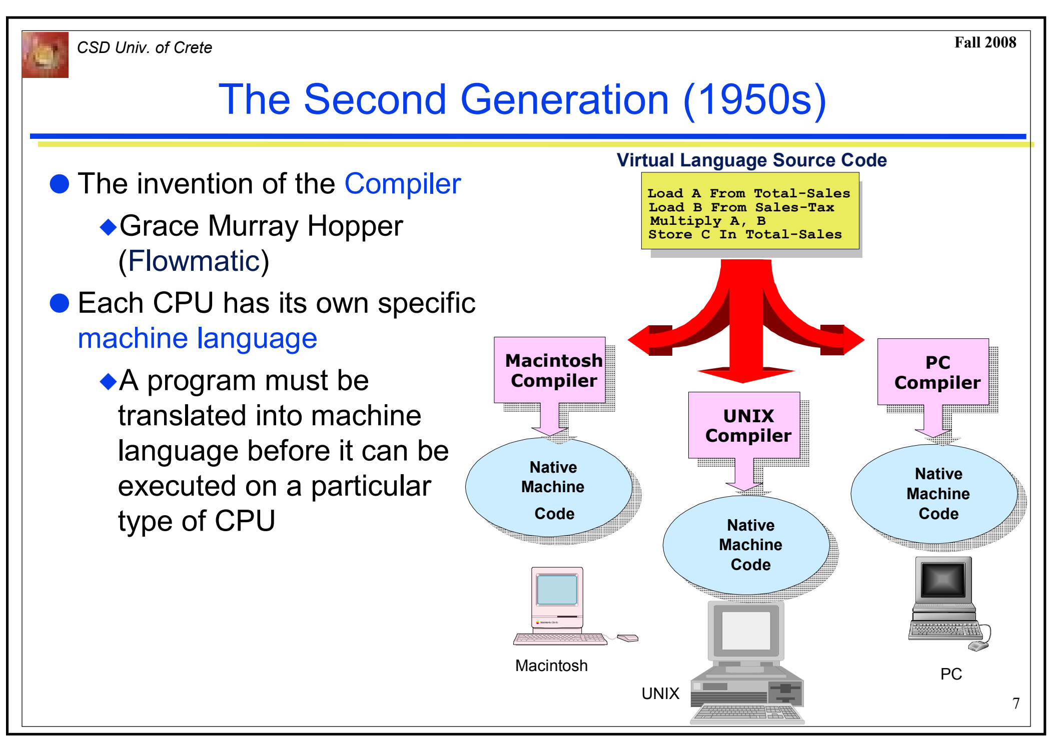

CSD Univ. of Crete
Fall 2008
Programming Language Timeline
FlowMatic
1955 Grace Hopper UNIVAC
ForTran
1956 John Backus IBM
AlgOL
1958 ACM Language Committee
LISP
1958 John McCarthy MIT
CoBOL
1960 Committee on Data Systems Languages
BASIC
1964 John Kemeny & Thomas Kurtz Dartmouth
PL/I
1964 IBM Committee
Simula
1967 Norwegian Computing Center Kristen Nygaard & Ole-Johan Dahl
Logo
1968 Seymour Papert MIT
Pascal
1970 Nicklaus Wirth Switzerland
C
1972 Dennis Ritchie & Kenneth Thompson Bell Labs
Smalltalk
1972 Alan Kay Xerox PARC
ADA
1981 DOD
Objective C
1985 Brad Cox Stepstone Systems
C++
1986 Bjarne Stroustrup Bell Labs
Eiffel
1989 Bertrand Meyer France
Visual BASIC
1990 Microsoft
Delphi
1995 Borland
Object CoBOL
1995 MicroFocus
Java
1995 Sun Microsystems
2

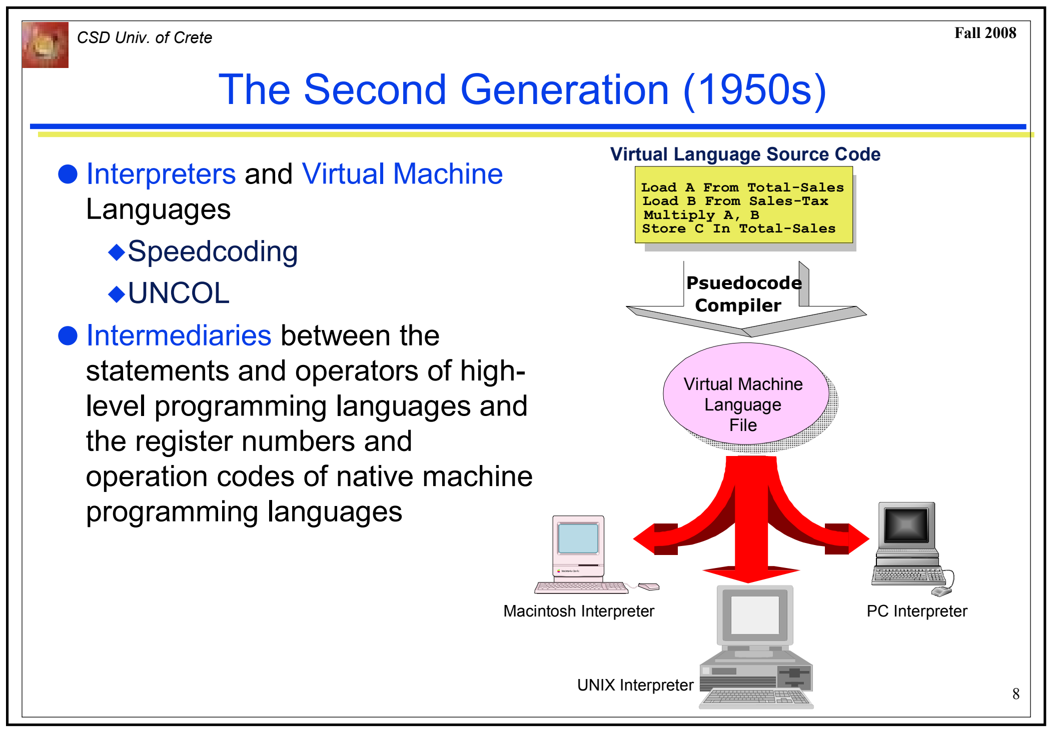

CSD Univ. of Crete
Fall 2008
Programming Language History
1960
1970
1980
1990
FORTRAN
Algol 60
COBOL
Lisp
Simula 67
PL/1
Algol 68
Prolog
Smalltalk 72
C
Pascal
Clu
Modula-2
Smalltalk 80
Ada
Oberon
Objective C
C++
Eiffel
Self
Modula-3
ANSI C++
Java
Ada 95
3

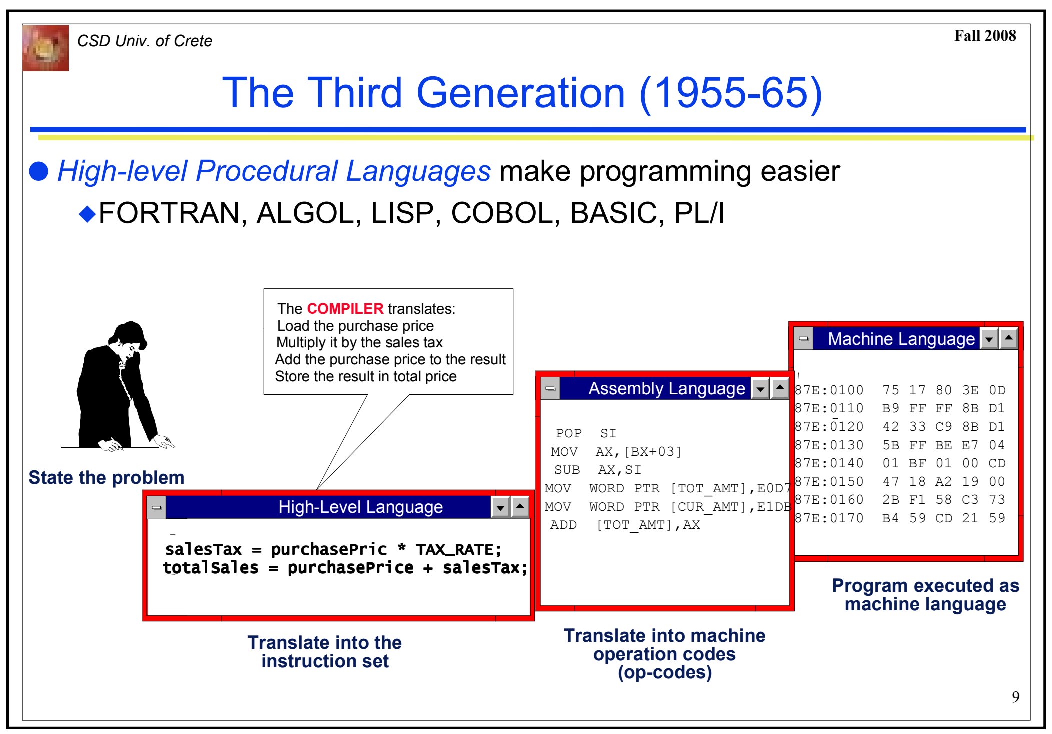

CSD Univ. of Crete
Fall 2008
Five Generations of Programming Languages
First Machine Languages
machine codes
Second Assembly Languages
symbolic assemblers
Third High Level Procedural Languages
(machine independent) imperative languages
Fourth Non-procedural Languages
domain specific application generators
Fifth Natural Languages
Each generation is at a higher level of abstraction
4

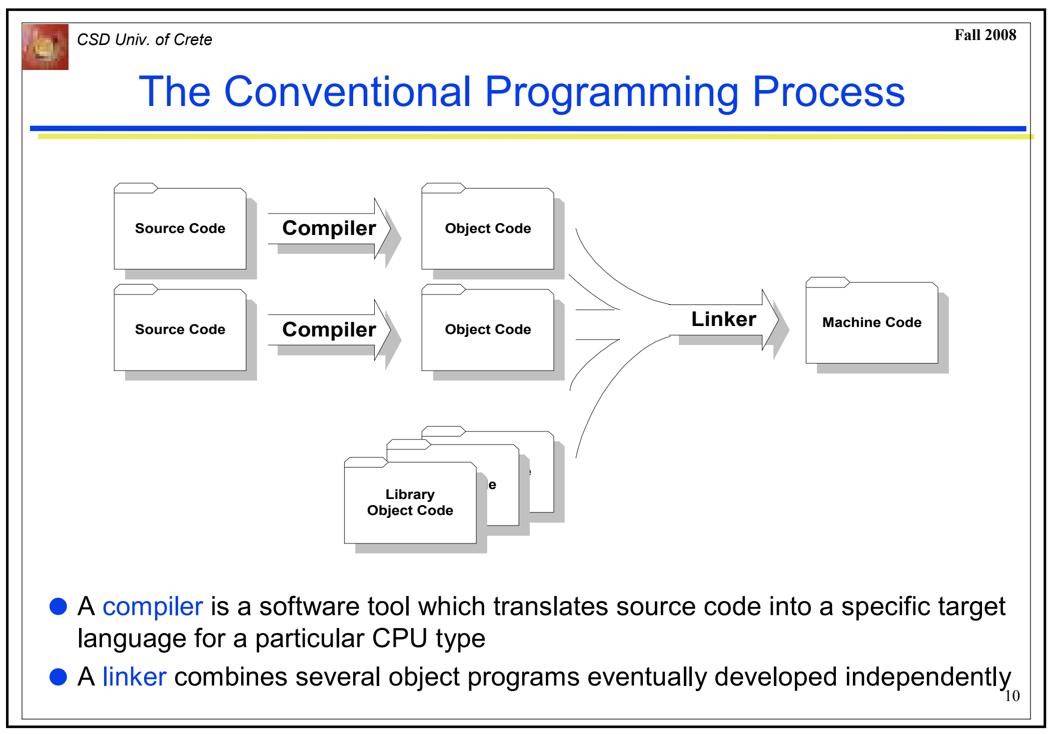

CSD Univ. of Crete
Fall 2008
The First Generation (1940s)
In the beginning … was the Stone Age: Machine Languages
Binary instruction strings
Introduced with the first programmable computer
Hardware dependent
I need to calculate the total sales. The sales tax rate is 10%. To write this program, I'll multiply the purchase price by the tax-rate and add the purchase price to the result. I'll store the result in the total sales field.
I need to know: What is the instruction to load from memory? Where is purchase price stored? What is the instruction to multiply? What do I multiply by? What is the instruction to add from memory? What is the instruction to store back into memory?
I need to: Load the purchase price Multiply it by the sales tax Add the purchase price to the result Store the result in total price
Machine Language
187E:0100 75 17 80 3E 0D
187E:0110 B9 FF FF 8B D1
187E:0120 42 33 C9 8B D1
187E:0130 5B FF BE E7 04
187E:0140 01 BF 01 00 CD
187E:0150 47 18 A2 19 00
187E:0160 2B F1 58 C3 73
187E:0170 B4 59 CD 21 59
State the problem
Translate into the instruction set
Translate into machine operation codes (op-codes)
Program entered and executed as machine language
5


CSD Univ. of Crete
Fall 2008
The Second Generation (Early 1950s)
Then we begin to study improvements: Assembly Languages
1-to-1 substitution of mnemonics for machine language commands
Hardware Dependent
I need to calculate the total sales. The sales tax rate is 10%. To write this program, I'll multiply the purchase price by the tax-rate and add the purchase price to the result. I'll store the result in the total sales field.
The ASSEMBLER converts instructions to op-codes: What is the instruction to load from memory? Where is purchase price stored? What is the instruction to multiply? What do I multiply by? What is the instruction to add from memory? What is the instruction to store back into memory?
I need to: Load the purchase price Multiply it by the sales tax Add the purchase price to the result Store the result in total price
Assembly Language
POP SI
MOV AX,[BX+03]
SUB AX,SI
MOV WORD PTR [TOT_AMT],E0D7
MOV WORD PTR [CUR_AMT],E1DB
ADD [TOT_AMT],AX
Machine Language
187E:0100 75 17 80 3E 0D
187E:0110 B9 FF FF 8B D1
187E:0120 42 33 C9 8B D1
187E:0130 5B FF BE E7 04
187E:0140 01 BF 01 00 CD
187E:0150 47 18 A2 19 00
187E:0160 2B F1 58 C3 73
187E:0170 B4 59 CD 21 59
State the problem
Translate into the instruction set
Translate into machine operation codes (op-codes)
Program executed as machine language
6


CSD Univ. of Crete
Fall 2008
The Second Generation (1950s)
The invention of the Compiler
Grace Murray Hopper (Flowmatic)
Each CPU has its own specific machine language
A program must be translated into machine language before it can be executed on a particular type of CPU
Virtual Language Source Code
Load A From Total-Sales
Load B From Sales-Tax
Multiply A, B
Store C In Total-Sales
Macintosh Compiler
UNIX Compiler
PC Compiler
Native Machine Code
Native Machine Code
Native Machine Code
Macintosh
UNIX
PC
7


CSD Univ. of Crete
Fall 2008
The Second Generation (1950s)
Interpreters and Virtual Machine Languages
Speedcoding
UNCOL
Intermediaries between the statements and operators of high-level programming languages and the register numbers and operation codes of native machine programming languages
Virtual Language Source Code
Load A From Total-Sales
Load B From Sales-Tax
Multiply A, B
Store C In Total-Sales
Psuedocode Compiler
Virtual Machine Language File
Macintosh Interpreter
UNIX Interpreter
PC Interpreter
8





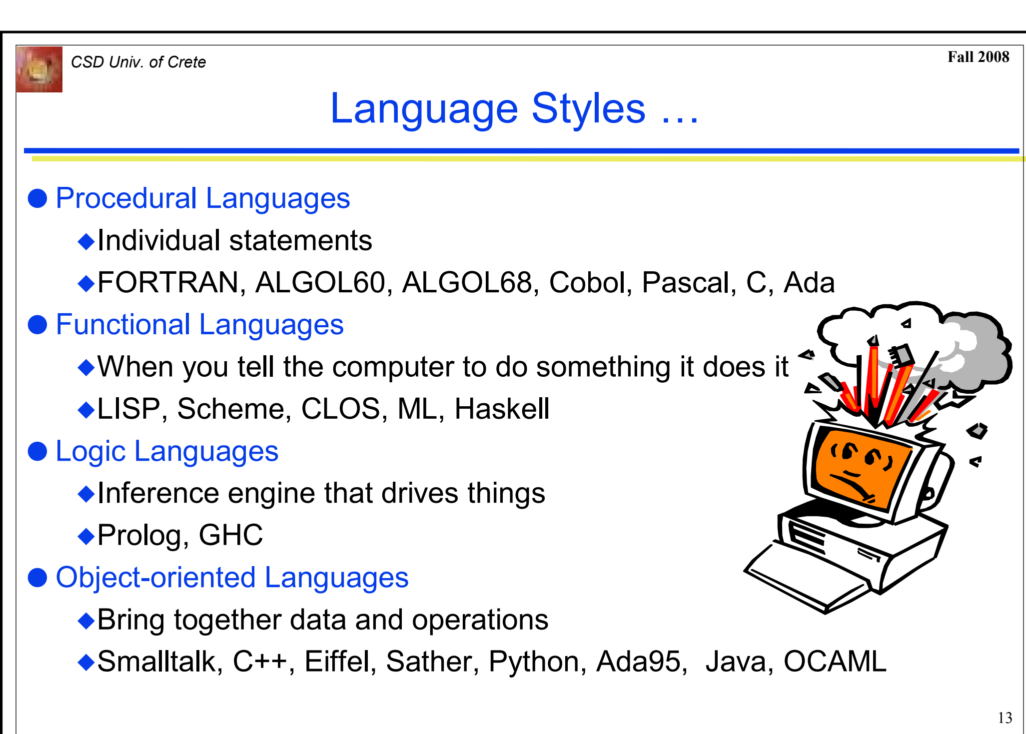
5

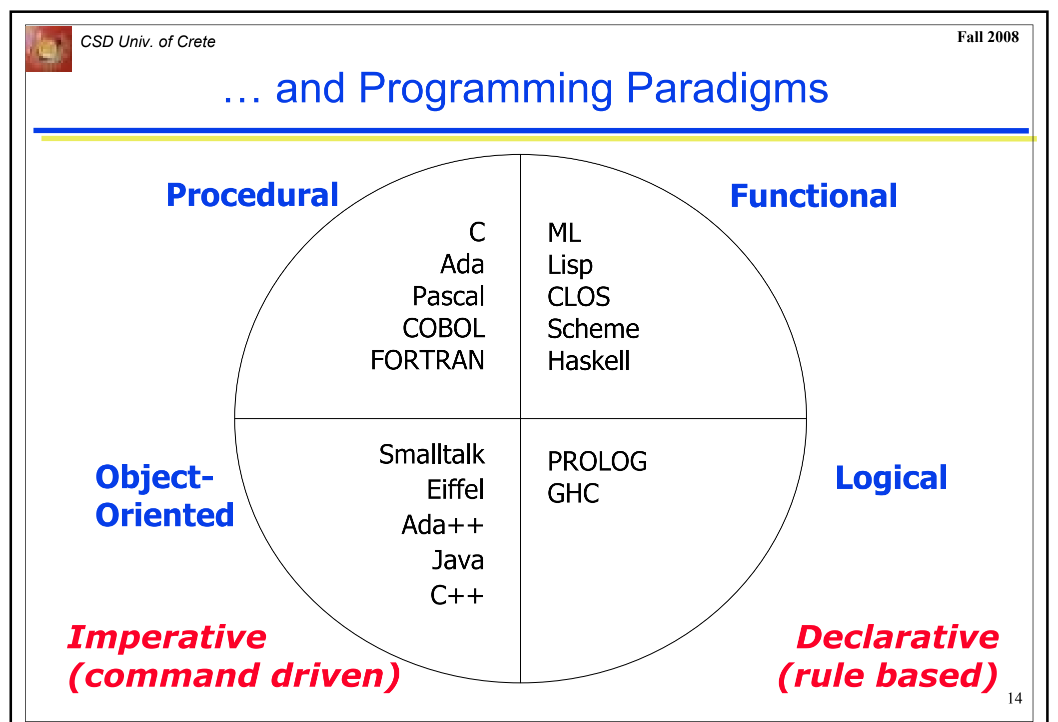
Fourth Generation Languages (1980)

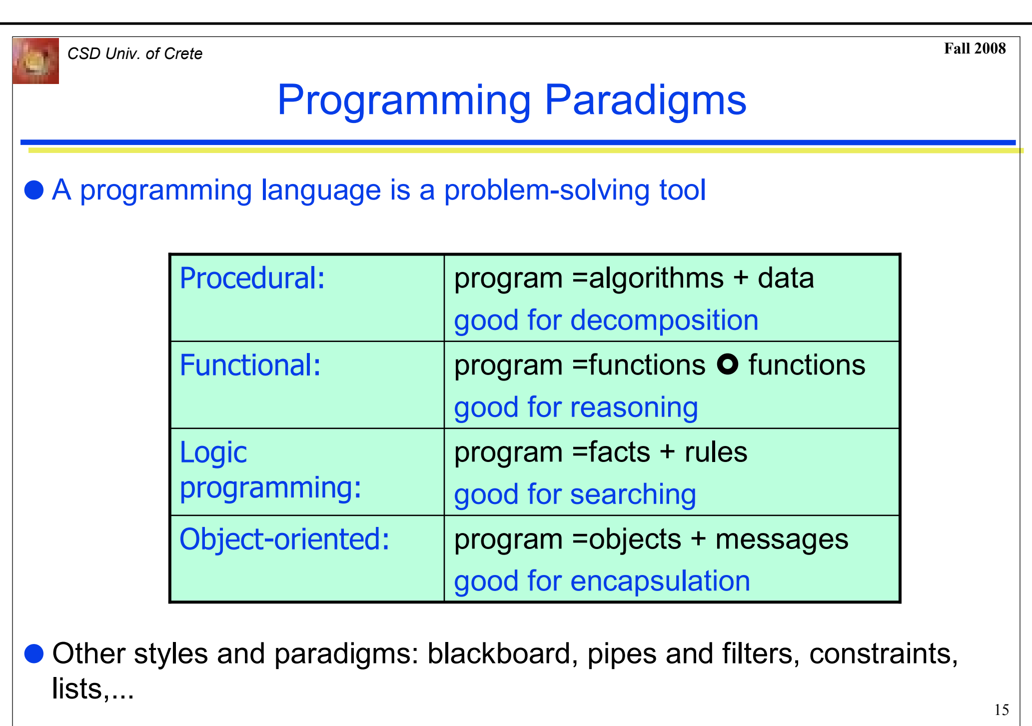
- Non-procedural Languages (problem-oriented)
  - User specifies what is to be done not how it is to be accomplished
  - Less user training is required
  - Designed to solve specific problems
- Diverse Types of 4GLs
  - Spreadsheet Languages
  - Database Query Languages
  - Decision Support Systems
  - Statistics
  - Simulation
  - Optimization
  - Decision Analysis
  - Presentation Graphics Systems

How do Programming Languages Differ?

- Common Constructs:
  - basic data types (numbers,etc.);
  - variables;
  - expressions;
  - statements;
  - keywords;
  - control constructs;
  - procedures;
  - comments;
  - errors ...
- Uncommon Constructs:
  - type declarations;
  - special types (strings, arrays, matrices,...);
  - sequential execution;
  - concurrency constructs;
  - packages/modules;
  - objects;
  - general functions;
  - generics;
  - modifiable state;...


CSD Univ. of Crete
Fall 2008
Language Styles …
Procedural Languages
Individual statements
FORTRAN, ALGOL60, ALGOL68, Cobol, Pascal, C, Ada
Functional Languages
When you tell the computer to do something it does it
LISP, Scheme, CLOS, ML, Haskell
Logic Languages
Inference engine that drives things
Prolog, GHC
Object-oriented Languages
Bring together data and operations
Smalltalk, C++, Eiffel, Sather, Python, Ada95, Java, OCAML
13


CSD Univ. of Crete
Fall 2008
… and Programming Paradigms
Procedural
C
Ada
Pascal
COBOL
FORTRAN
Functional
ML
Lisp
CLOS
Scheme
Haskell
Object-Oriented
Smalltalk
Eiffel
Ada++
Java
C++
Logical
PROLOG
GHC
Imperative (command driven)
Declarative (rule based)
14

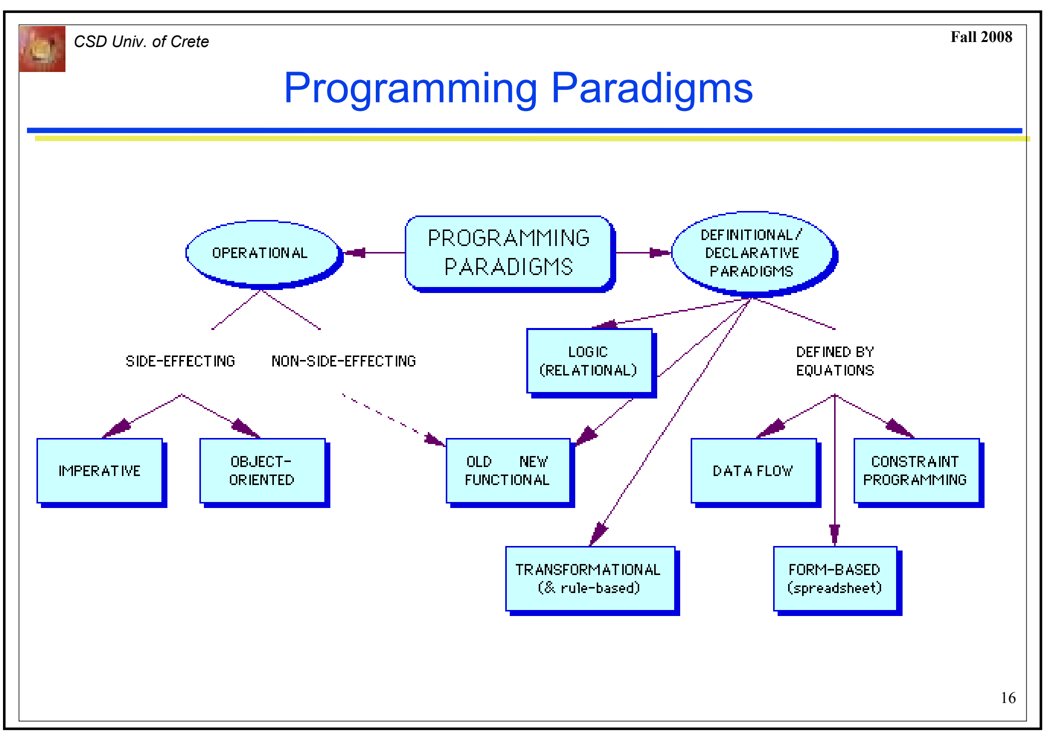


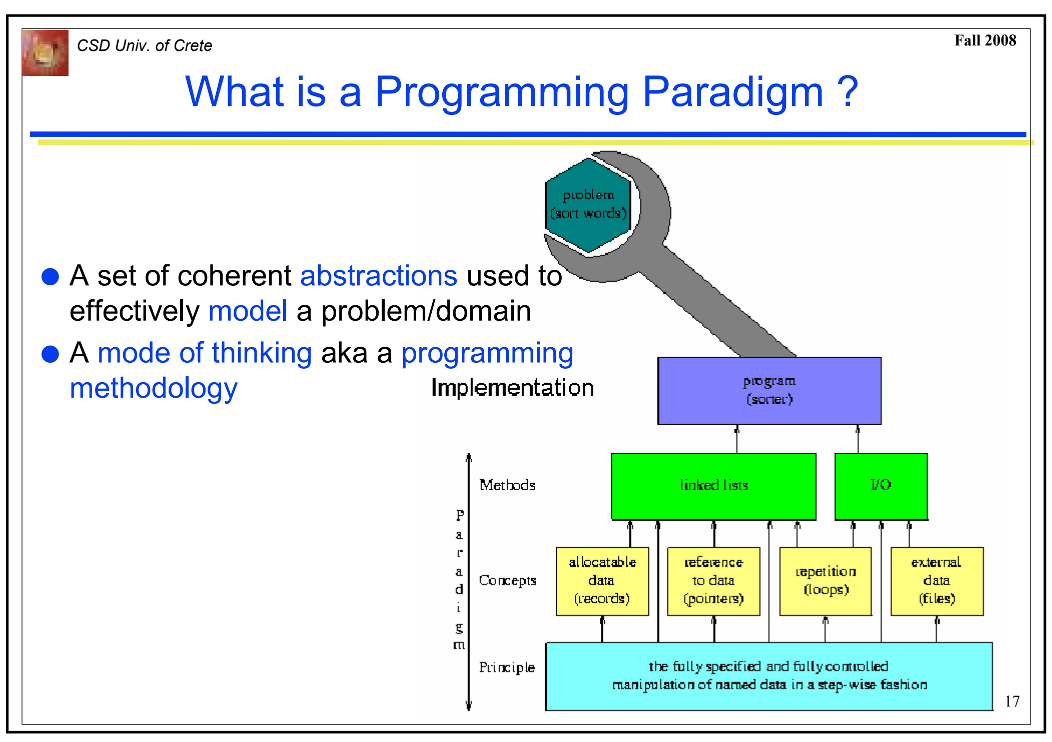
Programming Paradigms

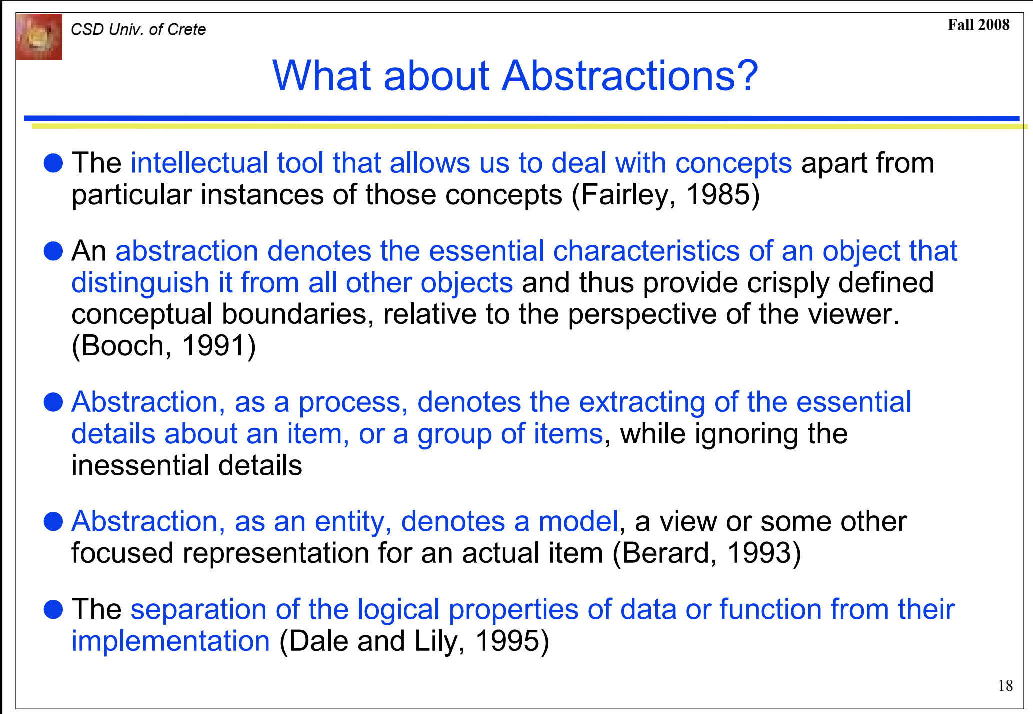
- A programming language is a problem-solving tool

Procedural:	program =algorithms + data good for decomposition
Functional:	program =functions o functions good for reasoning
Logic programming:	program =facts + rules good for searching
Object-oriented:	program =objects + messages good for encapsulation

- Other styles and paradigms: blackboard, pipes and filters, constraints, lists,...




CSD Univ. of Crete
Fall 2008
What is a Programming Paradigm ?
A set of coherent abstractions used to effectively model a problem/domain
A mode of thinking aka a programming methodology
problem (sort words)
Implementation
program (sorter)
Methods
linked lists
I/O
Paradigm
Concepts
allocatable data (records)
reference to data (pointers)
repetition (loops)
external data (files)
Principle
the fully specified and fully controlled manipulation of named data in a step-wise fashion
17

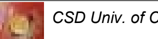

CSD Univ. of Crete
Fall 2008
What about Abstractions?
The intellectual tool that allows us to deal with concepts apart from particular instances of those concepts (Fairley, 1985)
An abstraction denotes the essential characteristics of an object that distinguish it from all other objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. (Booch, 1991)
Abstraction, as a process, denotes the extracting of the essential details about an item, or a group of items, while ignoring the inessential details
Abstraction, as an entity, denotes a model, a view or some other focused representation for an actual item (Berard, 1993)
The separation of the logical properties of data or function from their implementation (Dale and Lily, 1995)
18

CSD Univ. of Crete Fall 2008

What about Abstractions?

- In summary, abstraction allows us access to the relevant information regarding a problem/domain, and ignores the remainder
- Abstraction is a technique to manage, and cope with, the complexity of the tasks we perform
  - The ability to model at the right level a problem/domain, while ignore the rest
- The use of abstraction, both as a noun and a verb, allows us to
  - control the level and amount of detail,
  - communicate effectively with customers and users

19



CSD Univ. of Crete Fall 2008

Mechanisms of Abstraction

- Abstraction by parameterization abstracts from the identity of the data by replacing them with parameters
  - Example: a function to square an integer
- Abstraction by specification abstracts from the implementation details to the behavior users can depend on.
  - Related terms: contract, interface
- The history of PLs is a long road towards richer abstraction forms

20

Examples of Abstractions in PLs

- Procedural (abstraction of a statement) allows us to introduce new operations
  - Using the name of a sequence of instructions in place of the sequence of instructions
  - Parameterization allows high level of flexibility in the performance of operations
- Data (abstraction of a data type) allows us to introduce new types of data
  - A named collection that describes a data object
  - Provides a logical reference to the data object without concern for the underlying memory representation
- Control (abstraction of access details) allows us e.g., to iterate over items without knowing how the items are stored or obtained
  - A way of indicating the desired effect without establishing the actual control mechanism
  - Allows designers to model iteration (e.g., Iterator), concurrency, and synchronization

Examples of Abstractions

- Procedural

```
int function search(ListTYPE inList; int item)
double function square(int x)
void function sort(ListTYPE ioList)
```

- Data

```
public abstract class Employee implements Serializable
{ private Name name;
  private Address address;
  private String ssn="999999999";
  private String gender="female";
  private String maritalStatus="single";}
```

- Control

```
#('name' 32 (1/2)) do: [:value|value printOn: Transcript]
#(9 12 6 14 35 67 18) select: [:value|value even]
Iterator y= x.iterator();
while (y.hasNext()) examine(y.next());
```

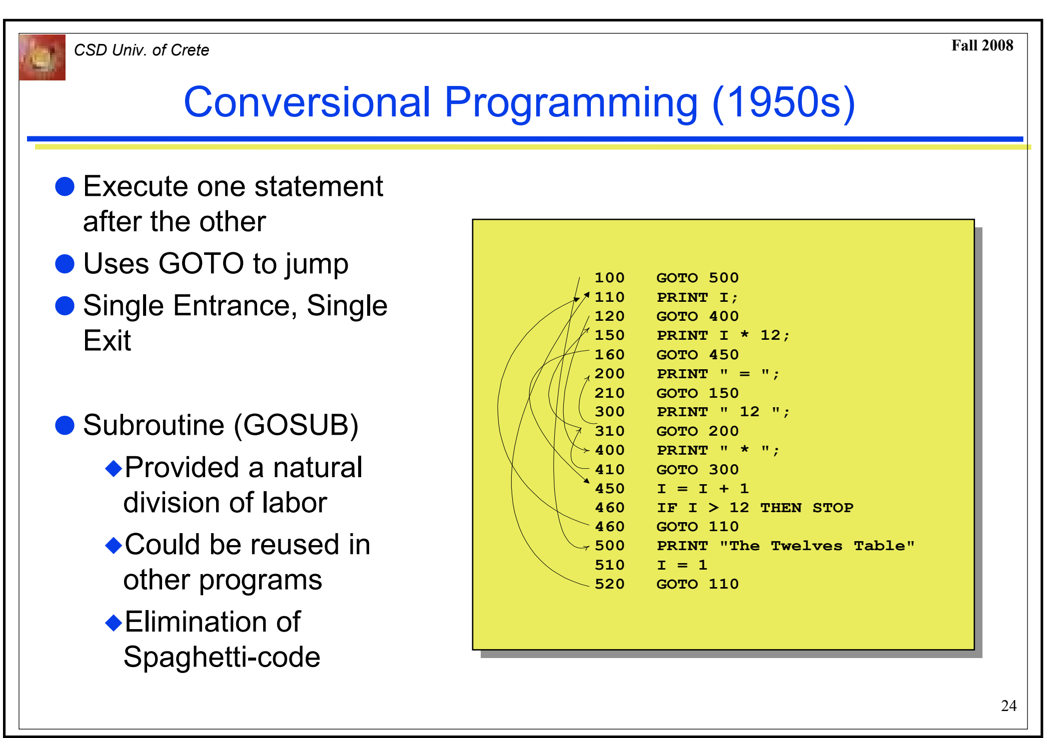
Programming Methodologies & Abstraction Concepts

Programming Methodologies	Abstraction Concepts	Programming Languages Constructs
Structured Programming	Explicit Control Structures	Do-while and other loops Blocks and so forth
Modular Programming	Information Hiding	Modules with well-defined interfaces
Abstract Data Types Programming	Data Representation Hiding	User-defined Data Types
Object-Oriented Programming	Reusing Software Artifacts	Classes, Inheritance, Polymorphism

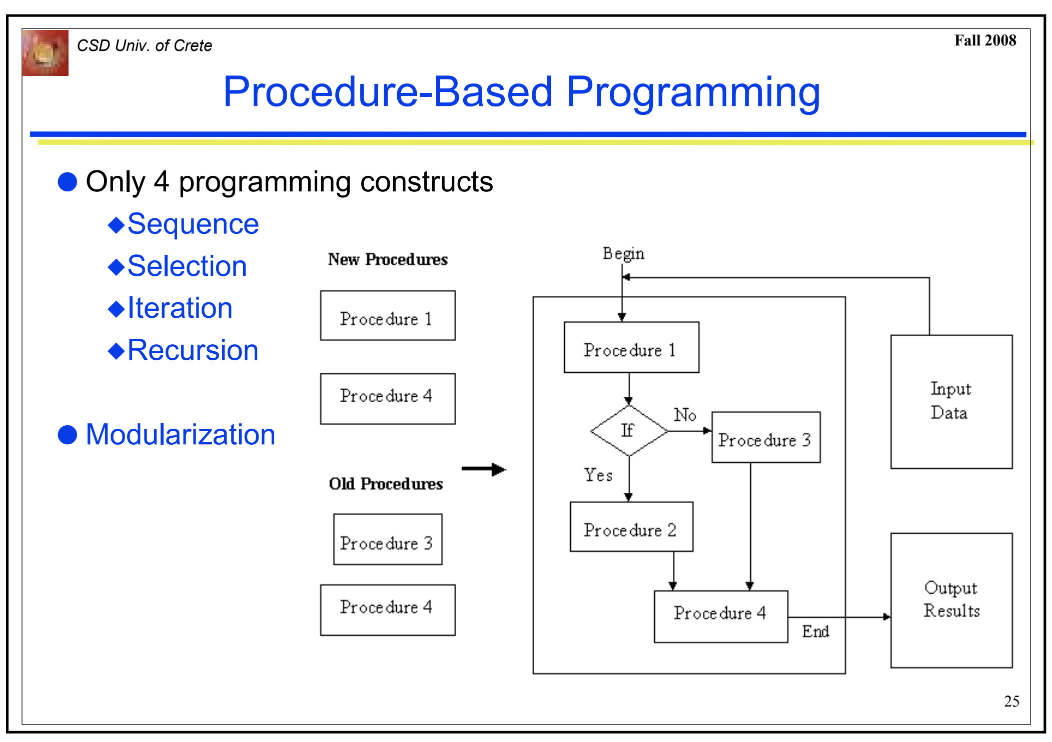
Conversional Programming (1950s)

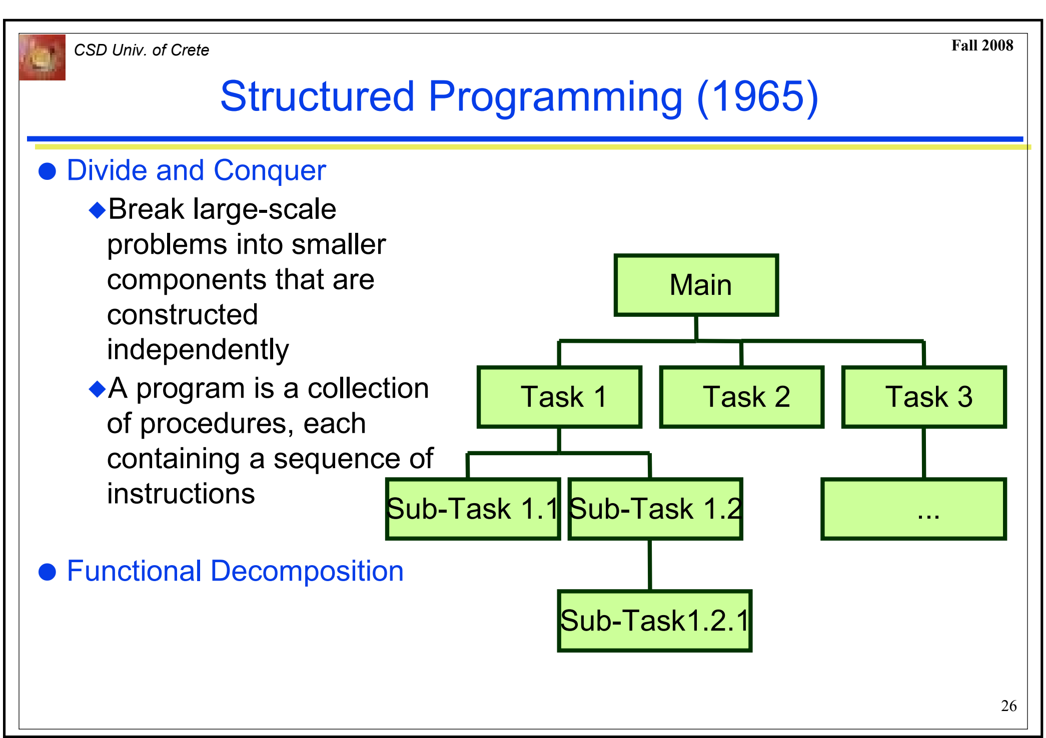
- Execute one statement after the other
- Uses GOTO to jump
- Single Entrance, Single Exit

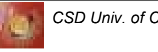
- Subroutine (GOSUB)
  - Provided a natural division of labor
  - Could be reused in other programs
  - Elimination of Spaghetti-code



```
100   GOTO 500
110   PRINT I;
120   GOTO 400
150   PRINT I * 12;
160   GOTO 450
200   PRINT " = ";
210   GOTO 150
300   PRINT " 12 ";
310   GOTO 200
400   PRINT " * ";
410   GOTO 300
450   I = I + 1
460   IF I > 12 THEN STOP
460   GOTO 110
500   PRINT "The Twelves Table"
510   I = 1
520   GOTO 110
```


CSD Univ. of Crete
Fall 2008
Procedure-Based Programming
Only 4 programming constructs
Sequence
Selection
Iteration
Recursion
Modularization
New Procedures
Procedure 1
Procedure 4
Old Procedures
Procedure 3
Procedure 4
Begin
Procedure 1
If
No
Yes
Procedure 3
Procedure 2
Procedure 4
End
Input Data
Output Results
25


CSD Univ. of Crete
Fall 2008
Structured Programming (1965)
Divide and Conquer
Break large-scale problems into smaller components that are constructed independently
A program is a collection of procedures, each containing a sequence of instructions
Functional Decomposition
Main
Task 1
Task 2
Task 3
Sub-Task 1.1
Sub-Task 1.2
...
Sub-Task1.2.1
26

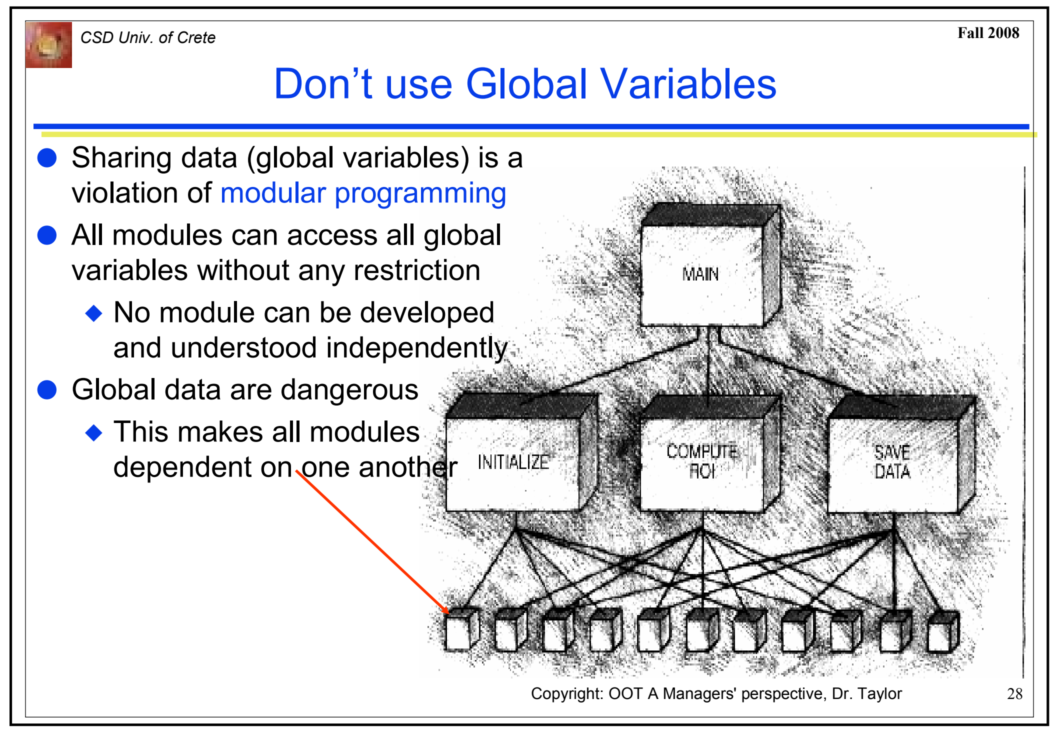


Structured Programming Problems

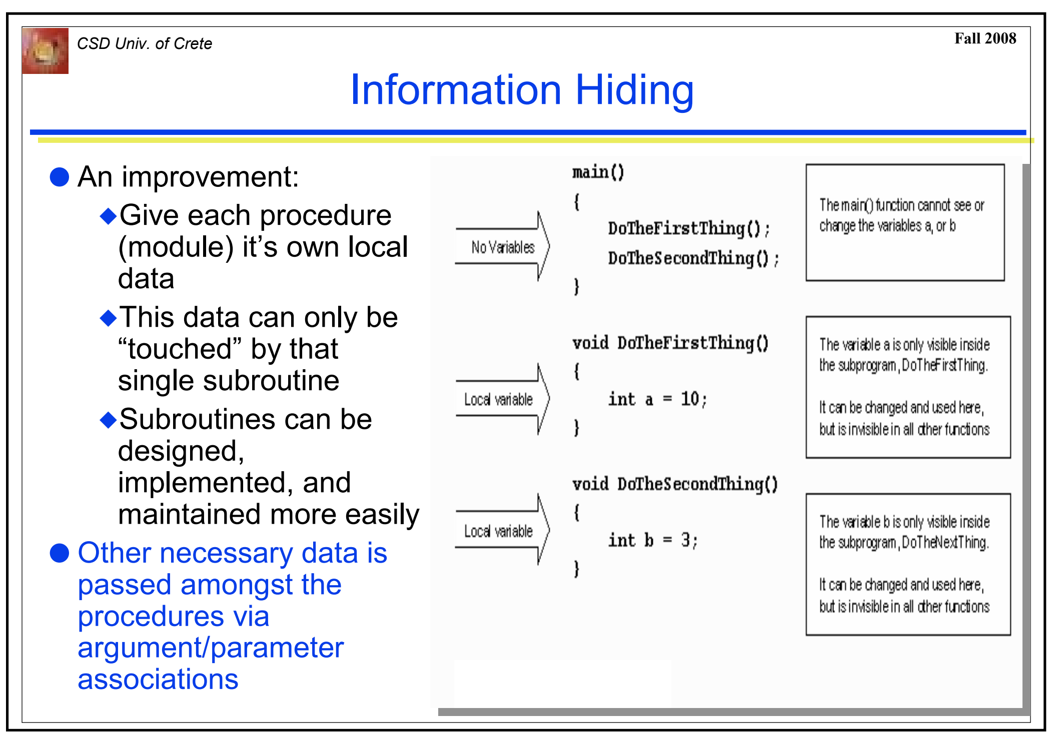
- Structured programming has a serious limitation:
  - It’s rarely possible to anticipate the design of a completed system before it’s implemented
  - The larger the system, the more restructuring takes place
- Software development had focused on the modularization of code
  - data moved around
    - argument/parameter associations
  - or data was global
    - works okay for tiny programs
    - Not so good when variables number in the hundreds

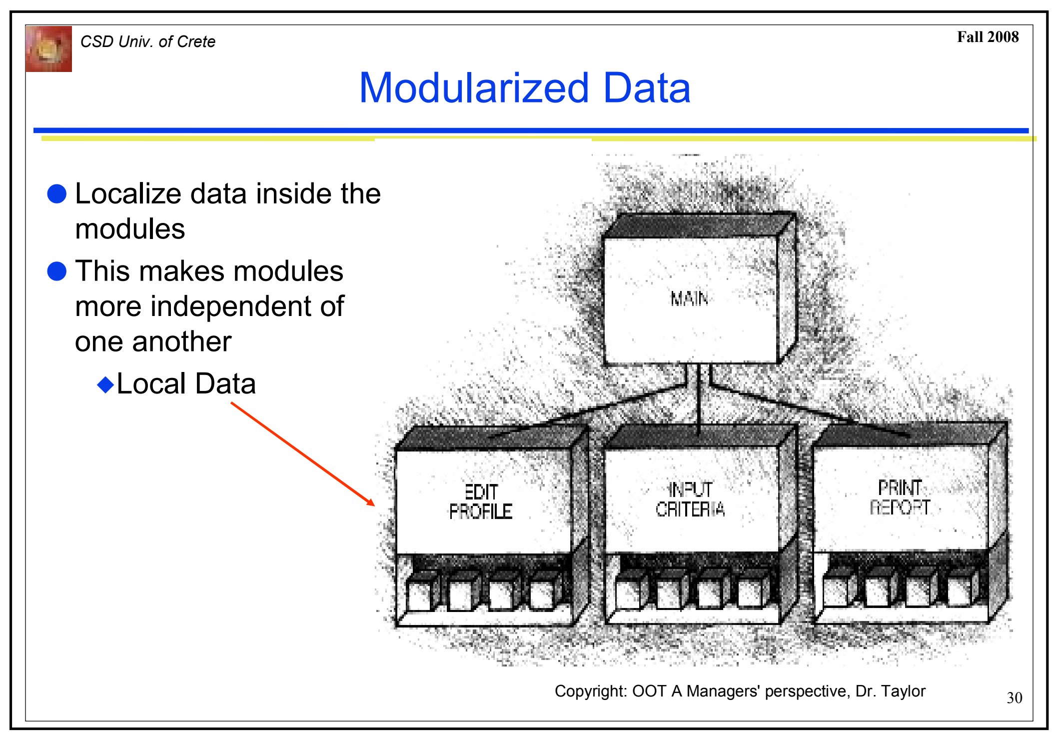
Don’t use Global Variables

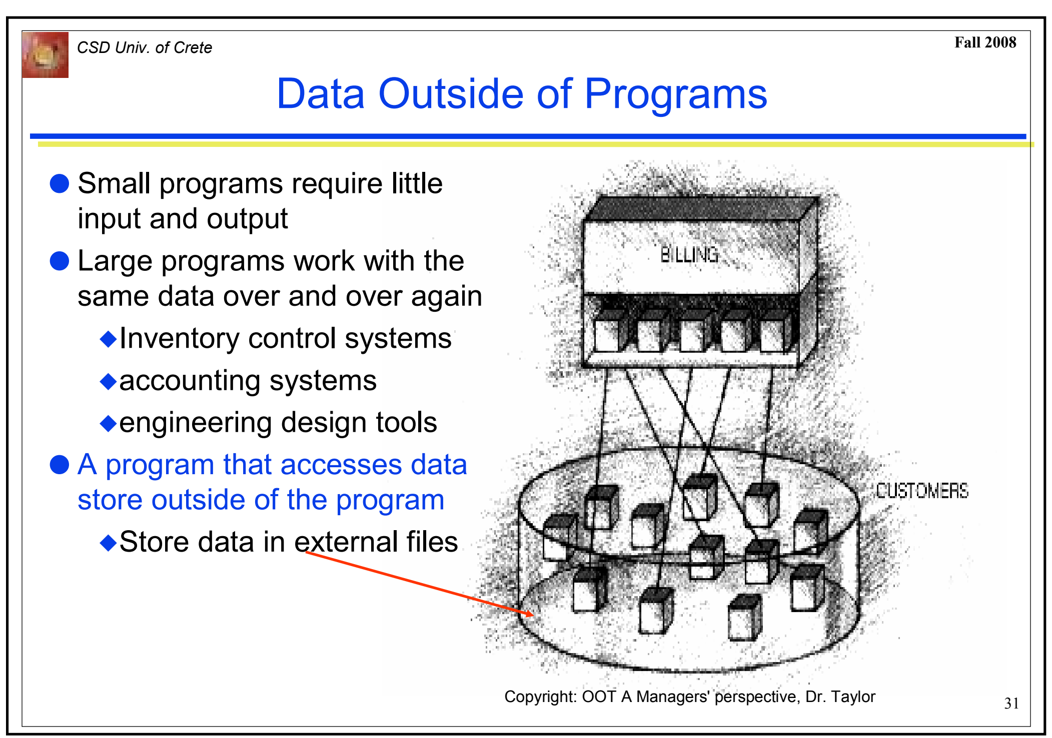
- Sharing data (global variables) is a violation of modular programming
- All modules can access all global variables without any restriction
  - No module can be developed and understood independently
- Global data are dangerous
  - This makes all modules dependent on one another

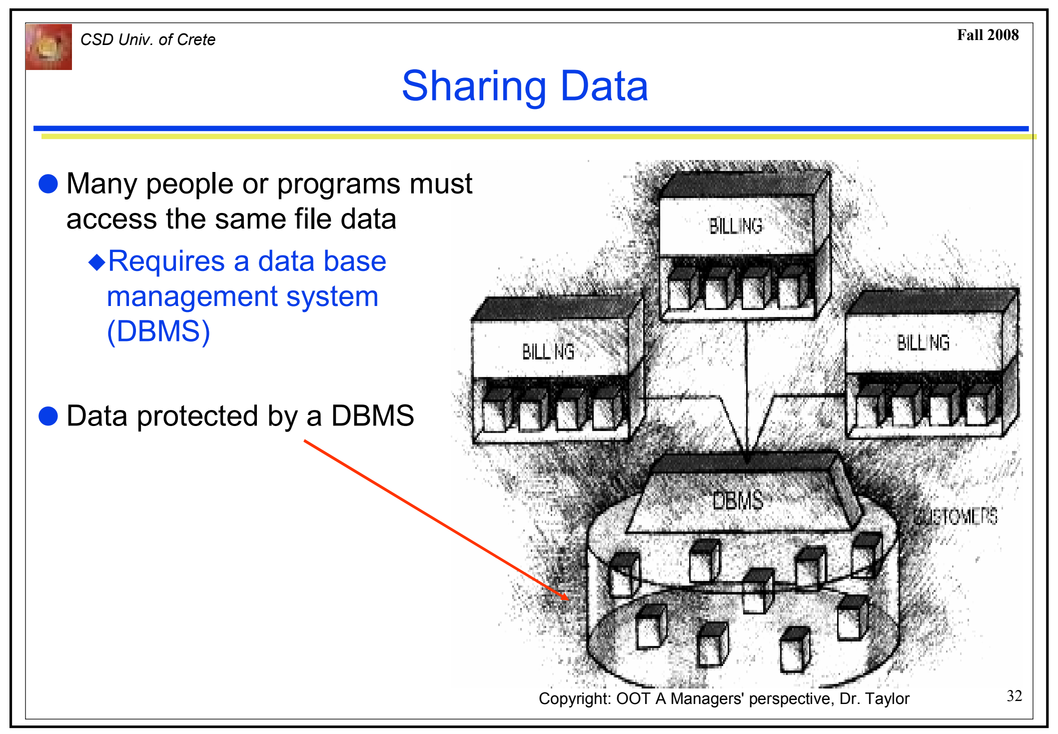


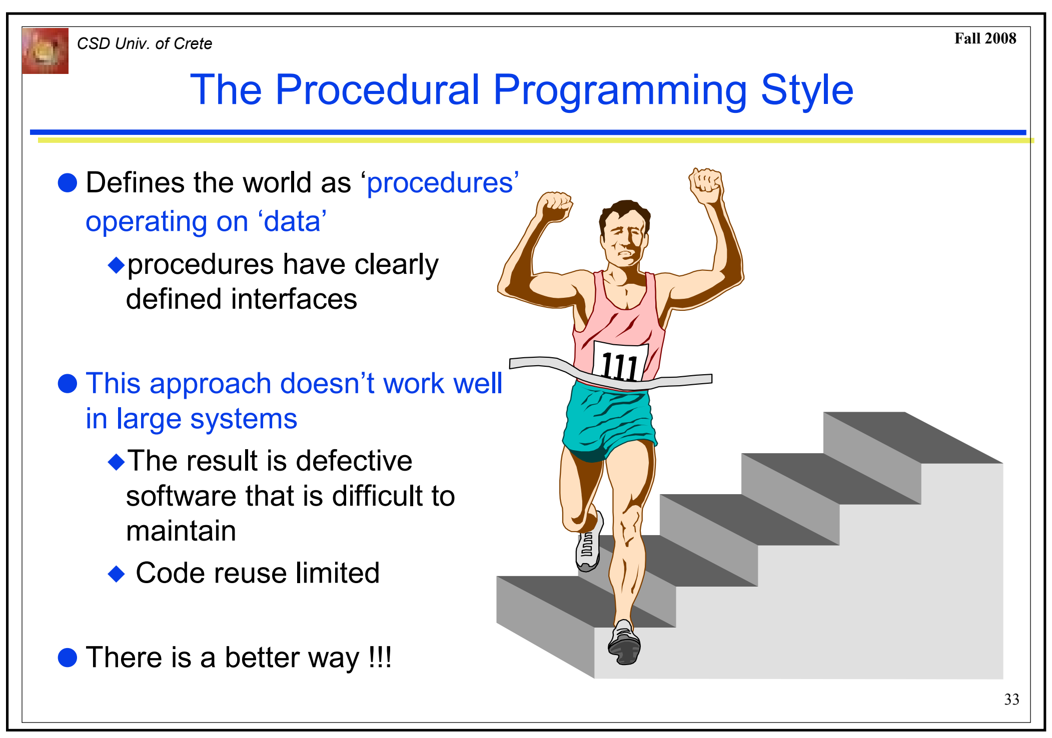
Copyright: OOT A Managers' perspective, Dr. Taylor

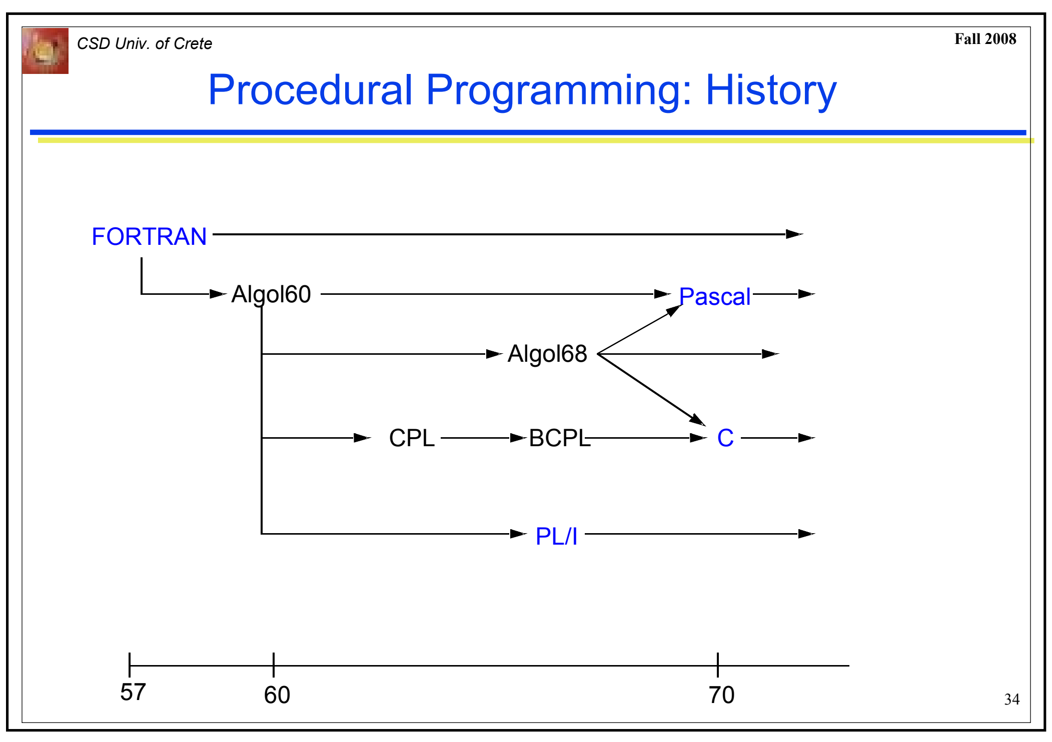

CSD Univ. of Crete
Fall 2008
Information Hiding
An improvement:
Give each procedure (module) it's own local data
This data can only be "touched" by that single subroutine
Subroutines can be designed, implemented, and maintained more easily
Other necessary data is passed amongst the procedures via argument/parameter associations
No Variables
Local variable
Local variable
main()
{
DoTheFirstThing();
DoTheSecondThing();
}
void DoTheFirstThing()
{
int a = 10;
}
void DoTheSecondThing()
{
int b = 3;
}
The main() function cannot see or change the variables a, or b
The variable a is only visible inside the subprogram, DoTheFirstThing.
It can be changed and used here, but is invisible in all other functions
The variable b is only visible inside the subprogram, DoTheNextThing.
It can be changed and used here, but is invisible in all other functions

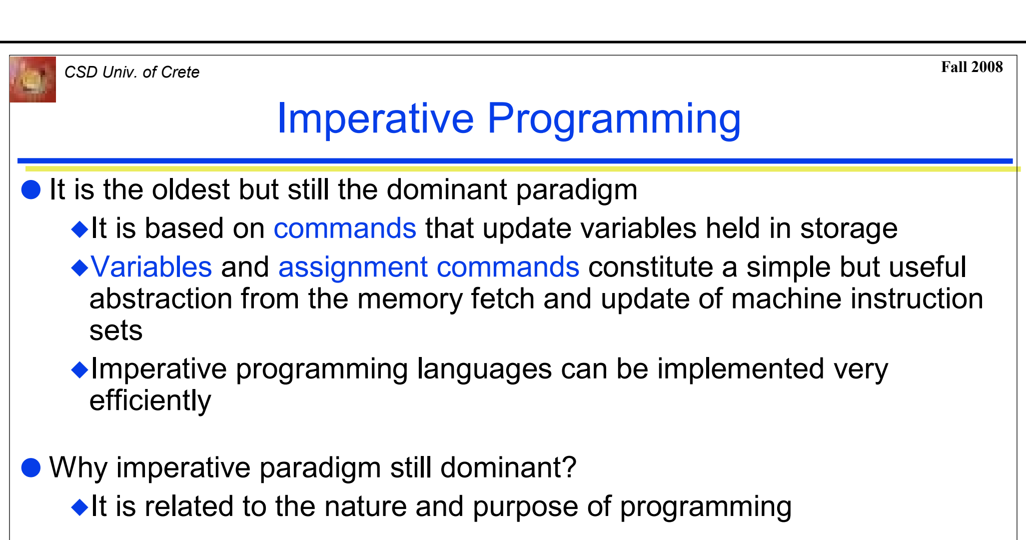

CSD Univ. of Crete
Fall 2008
Modularized Data
Localize data inside the modules
This makes modules more independent of one another
Local Data
MAIN
EDIT PROFILE
INPUT CRITERIA
PRINT REPORT
Copyright: OOT A Managers' perspective, Dr. Taylor
30

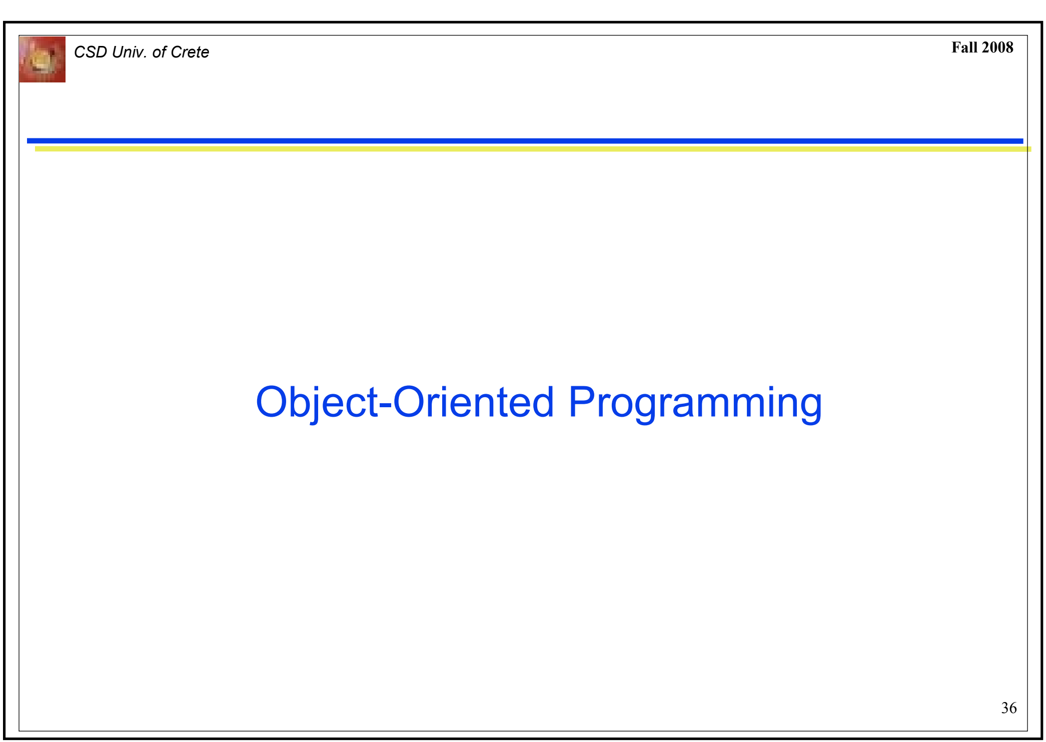

CSD Univ. of Crete
Fall 2008
Data Outside of Programs
Small programs require little input and output
Large programs work with the same data over and over again
Inventory control systems
accounting systems
engineering design tools
A program that accesses data store outside of the program
Store data in external files
BILLING
CUSTOMERS
Copyright: OOT A Managers' perspective, Dr. Taylor
31


CSD Univ. of Crete
Fall 2008
Sharing Data
Many people or programs must access the same file data
Requires a data base management system (DBMS)
Data protected by a DBMS
BILLING
BILLING
BILLING
DBMS
Copyright: OOT A Managers' perspective, Dr. Taylor
32

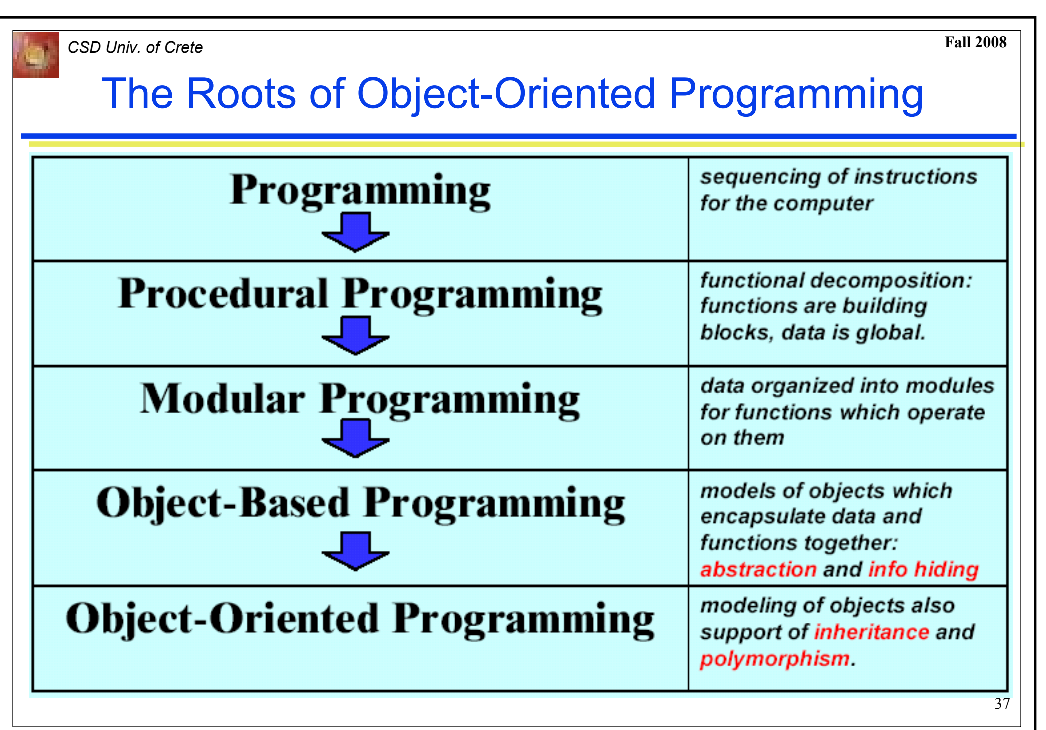

CSD Univ. of Crete
Fall 2008
The Procedural Programming Style
Defines the world as 'procedures' operating on 'data'
procedures have clearly defined interfaces
This approach doesn't work well in large systems
The result is defective software that is difficult to maintain
Code reuse limited
There is a better way !!!
111
33

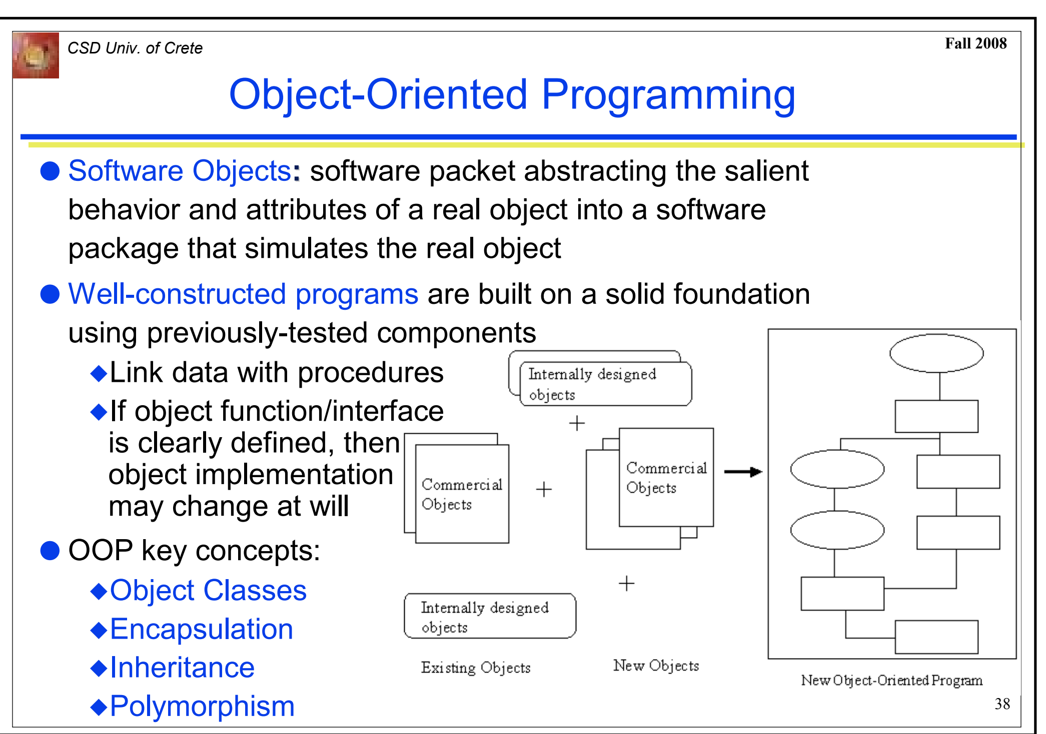

CSD Univ. of Crete
Fall 2008
Procedural Programming: History
FORTRAN
Algol60
Pascal
Algol68
CPL
BCPL
C
PL/I
57
60
70
34

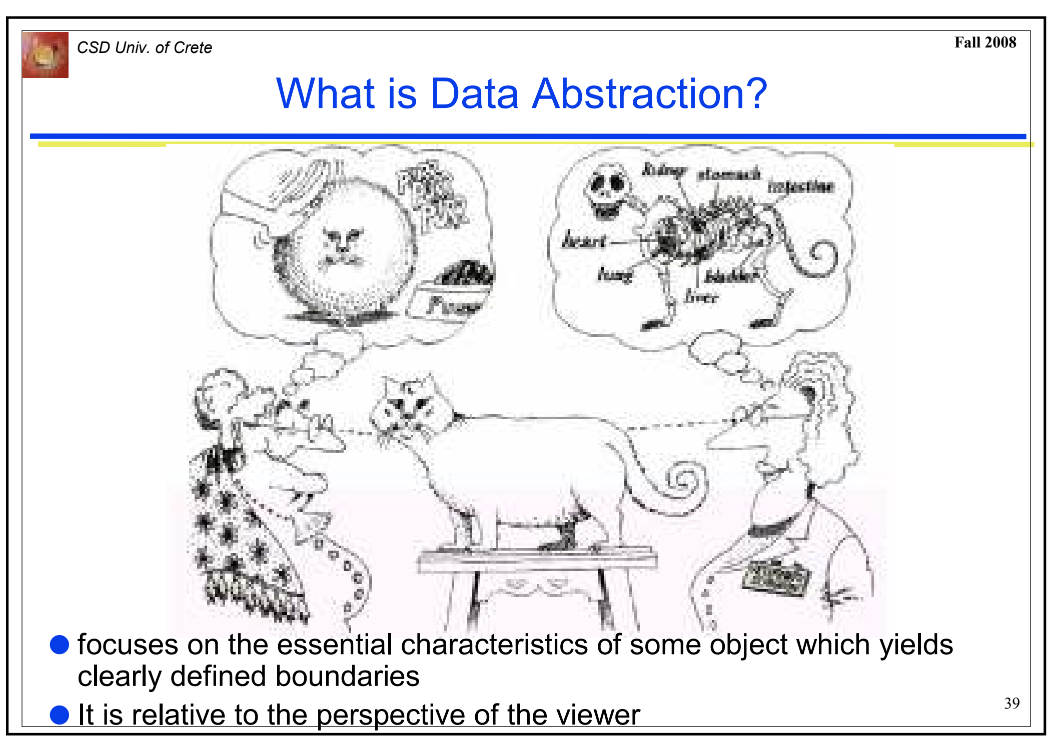


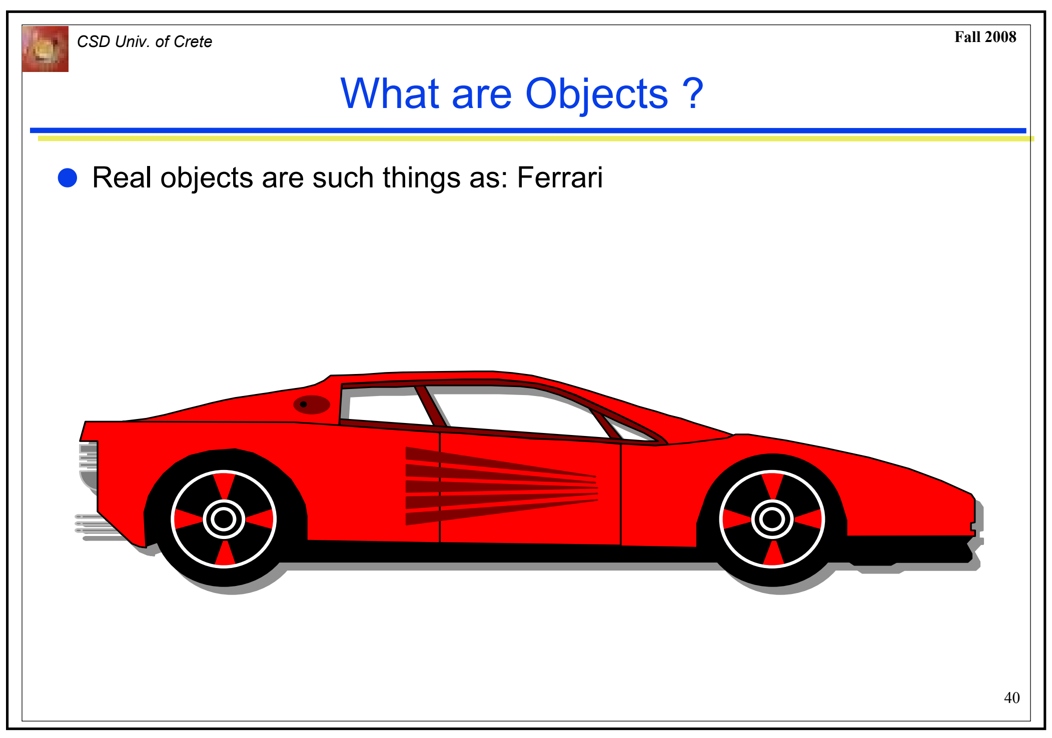


18

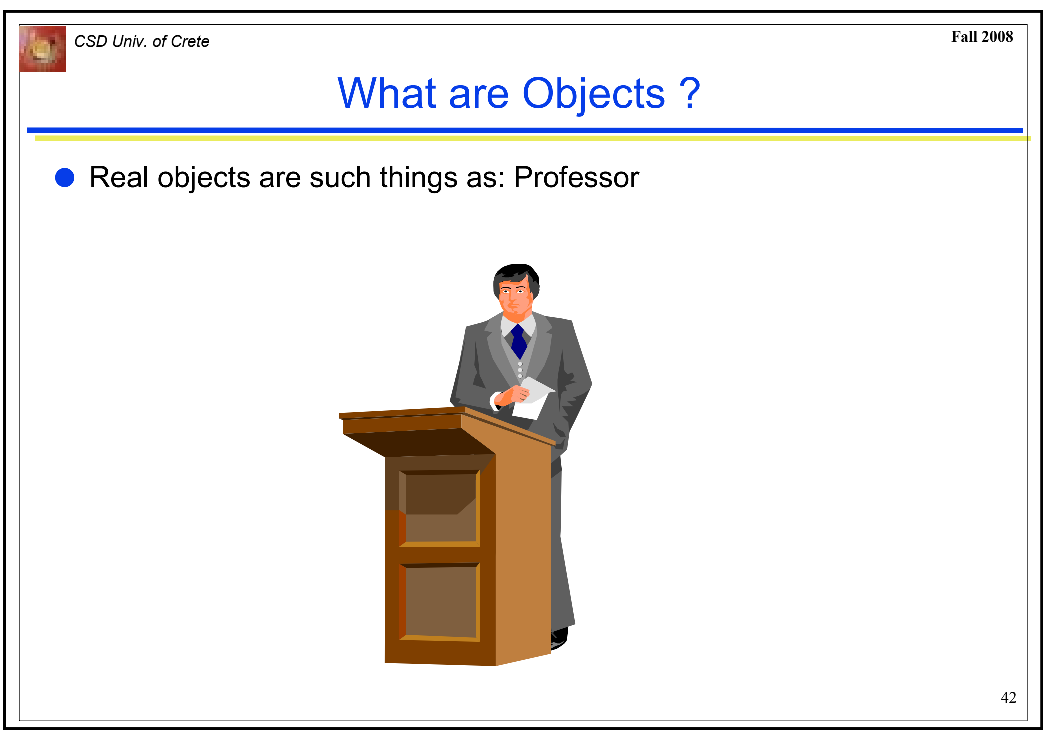

CSD Univ. of Crete
Fall 2008
The Roots of Object-Oriented Programming
Programming
sequencing of instructions for the computer
Procedural Programming
functional decomposition: functions are building blocks, data is global.
Modular Programming
data organized into modules for functions which operate on them
Object-Based Programming
models of objects which encapsulate data and functions together: abstraction and info hiding
Object-Oriented Programming
modeling of objects also support of inheritance and polymorphism.
37

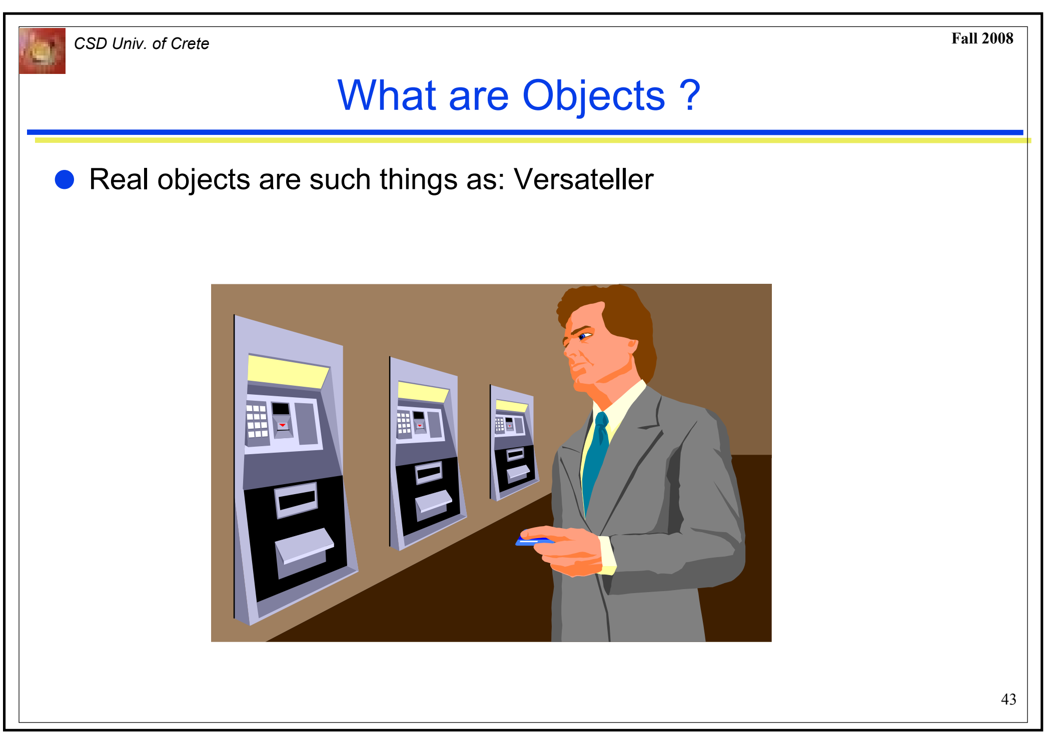

CSD Univ. of Crete
Fall 2008
Object-Oriented Programming
Software Objects: software packet abstracting the salient behavior and attributes of a real object into a software package that simulates the real object
Well-constructed programs are built on a solid foundation using previously-tested components
Link data with procedures
If object function/interface is clearly defined, then object implementation may change at will
OOP key concepts:
Object Classes
Encapsulation
Inheritance
Polymorphism
Internally designed objects
Commercial Objects
Commercial Objects
Internally designed objects
Existing Objects
New Objects
New Object-Oriented Program
38

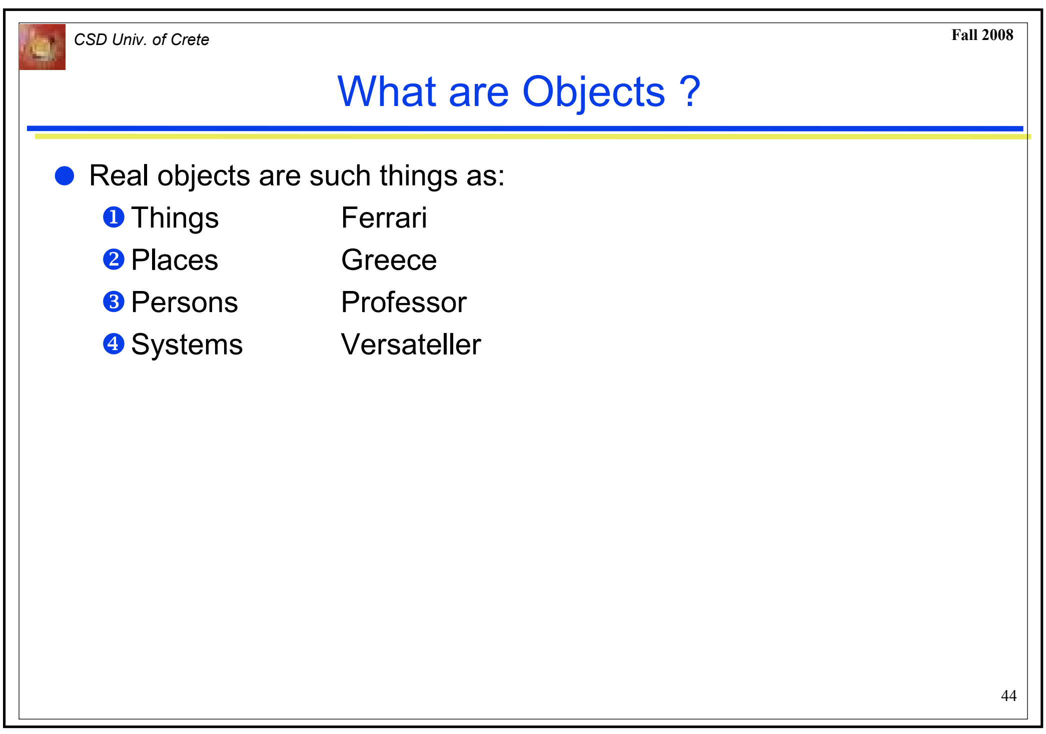

CSD Univ. of Crete
Fall 2008
What is Data Abstraction?
kidney
stomach
intestine
heart
lung
bladder
liver
focuses on the essential characteristics of some object which yields clearly defined boundaries
It is relative to the perspective of the viewer
39


CSD Univ. of Crete
Fall 2008
What are Objects ?
Real objects are such things as: Ferrari
40

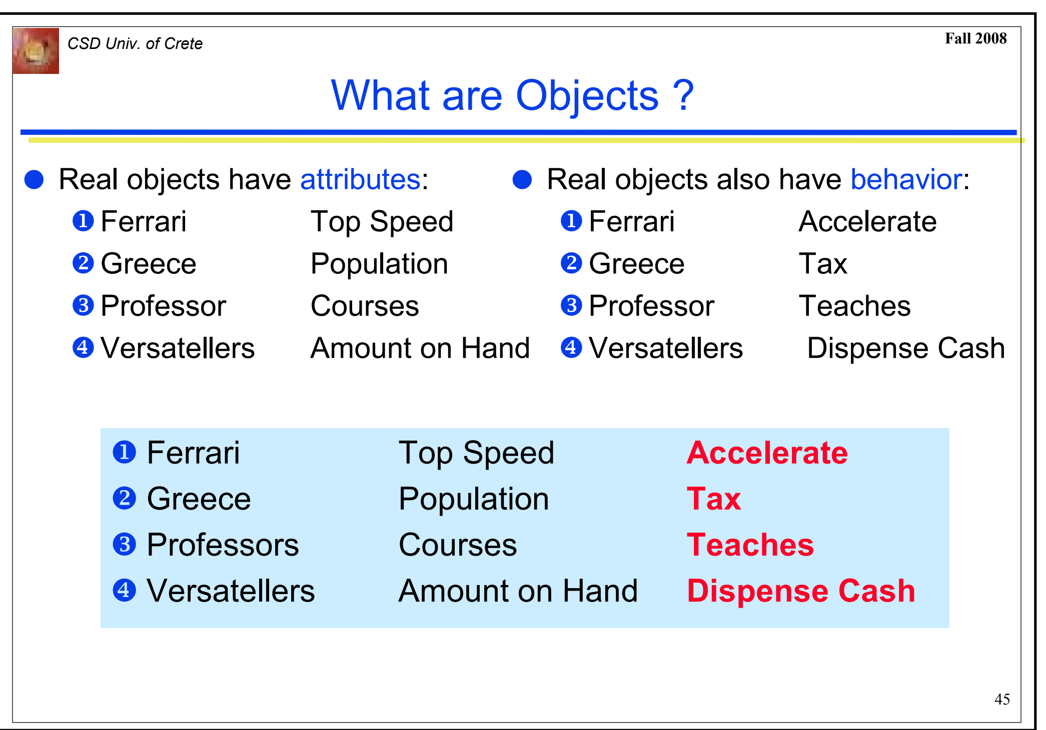
CSD Univ. of Crete
Fall 2008
What are Objects ?
Real objects are such things as: Greece
41

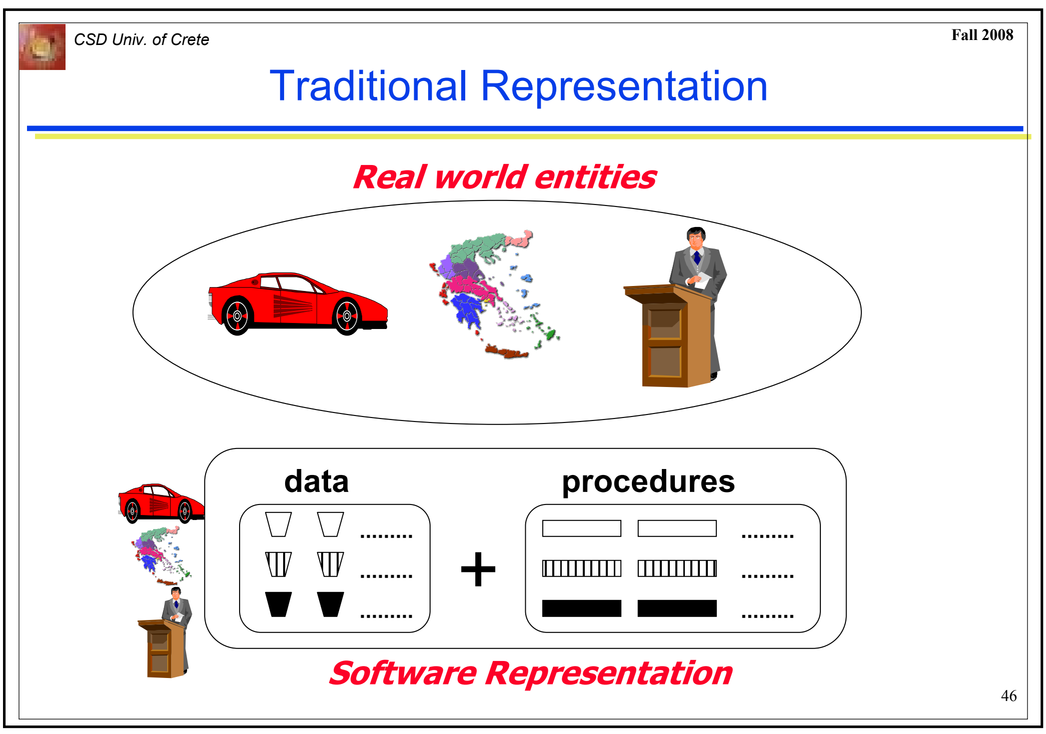

CSD Univ. of Crete
Fall 2008
What are Objects ?
Real objects are such things as: Professor
42

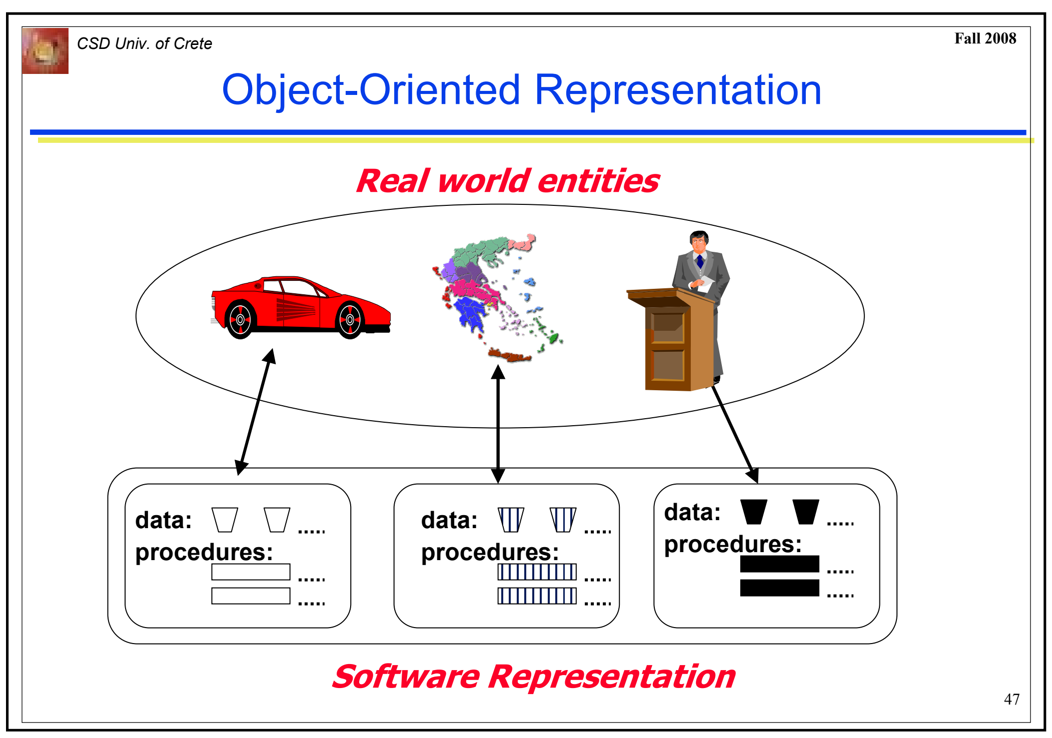


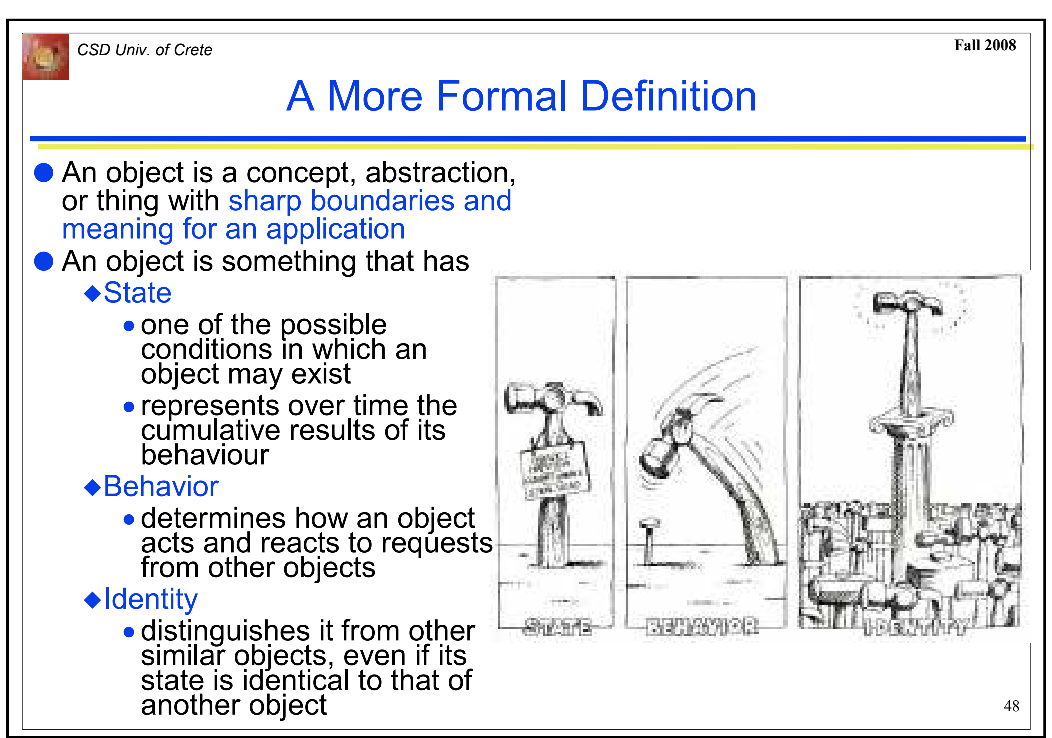


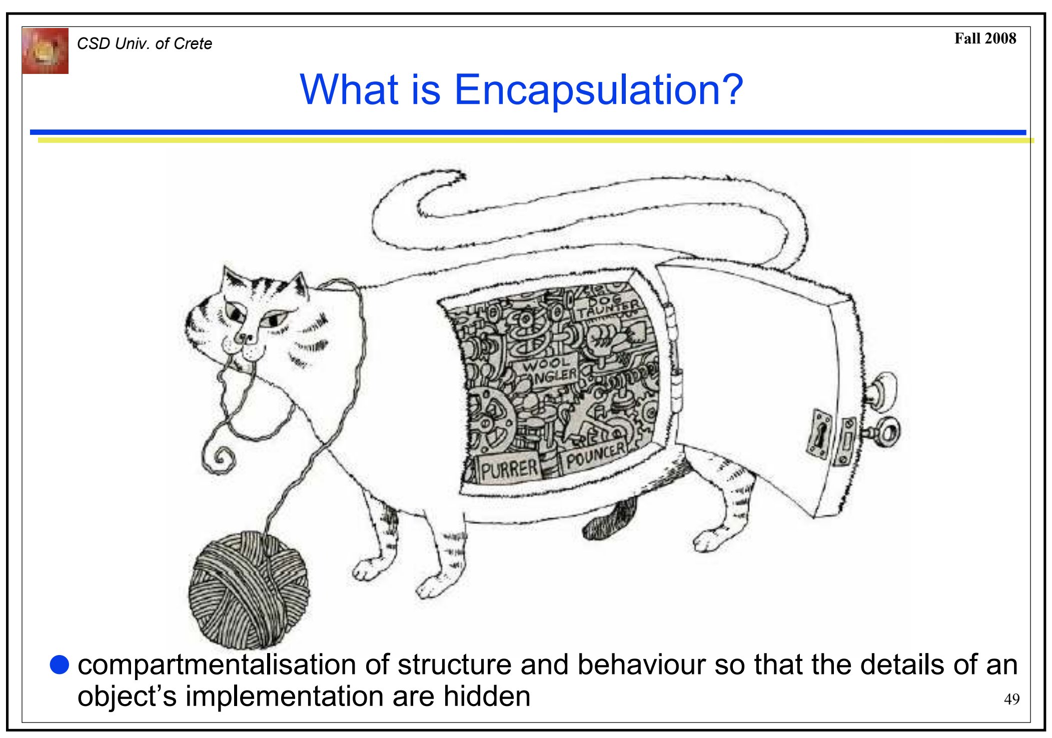
22

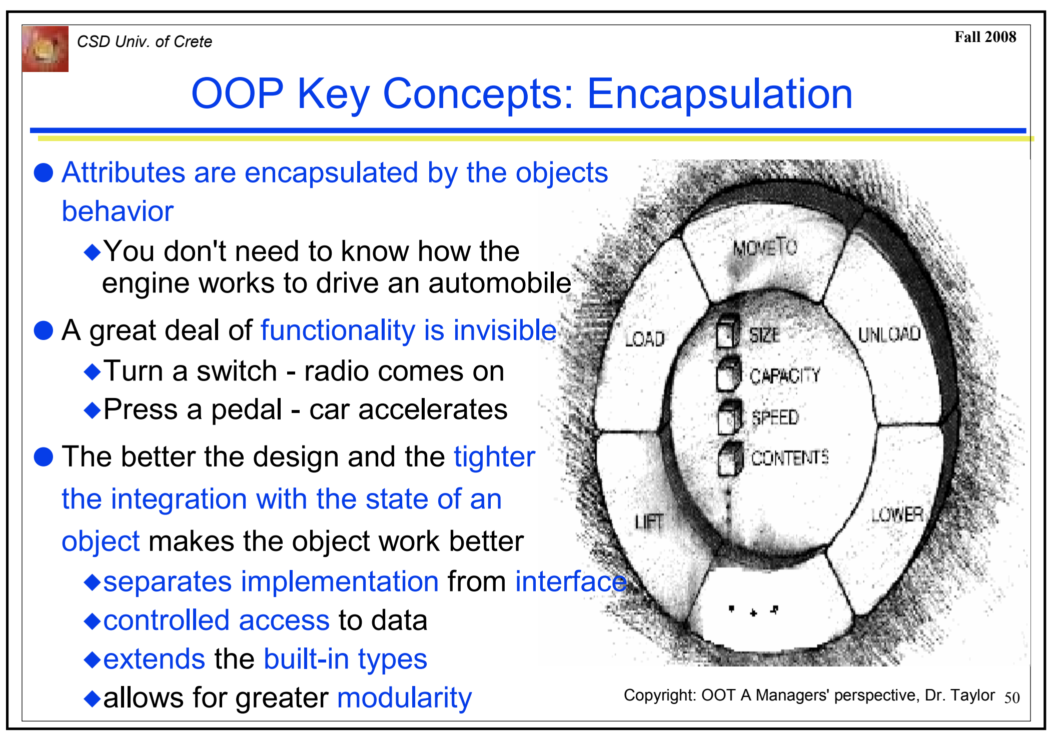

CSD Univ. of Crete
Fall 2008
What are Objects ?
Real objects have attributes:
❶ Ferrari Top Speed
❷ Greece Population
❸ Professor Courses
❹ Versatellers Amount on Hand
Real objects also have behavior:
❶ Ferrari Accelerate
❷ Greece Tax
❸ Professor Teaches
❹ Versatellers Dispense Cash
❶ Ferrari Top Speed Accelerate
❷ Greece Population Tax
❸ Professors Courses Teaches
❹ Versatellers Amount on Hand Dispense Cash
45

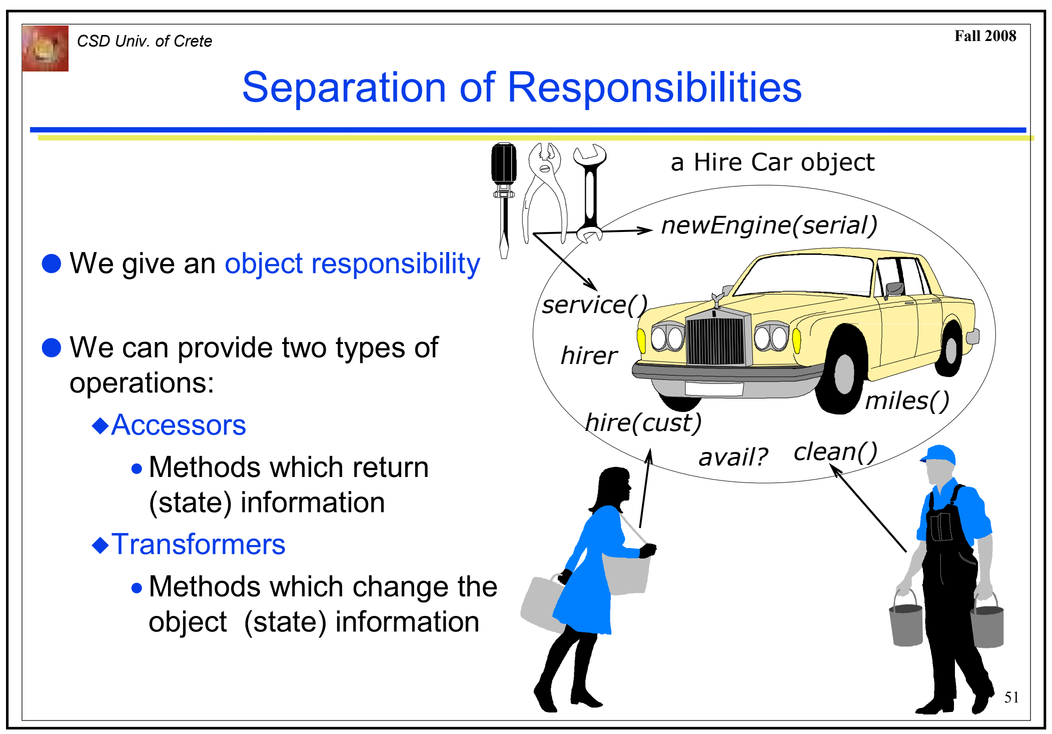

CSD Univ. of Crete
Fall 2008
Traditional Representation
Real world entities
data
procedures
+
Software Representation
46

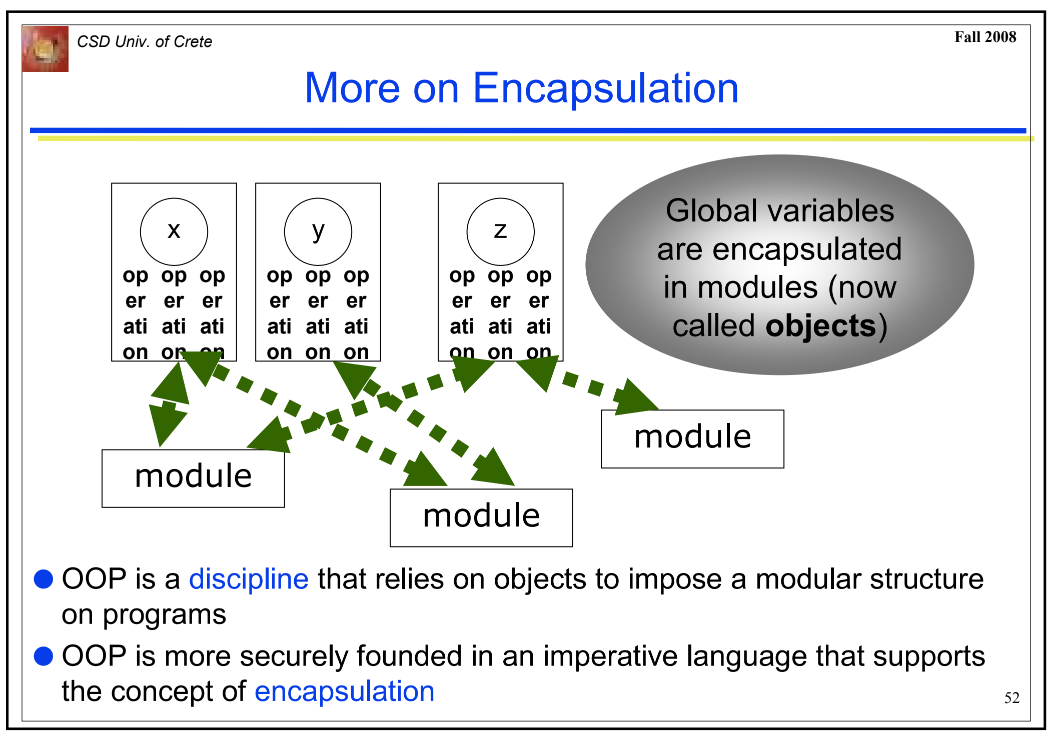

CSD Univ. of Crete
Fall 2008
Object-Oriented Representation
Real world entities
data:
procedures:
data:
procedures:
data:
procedures:
Software Representation
47

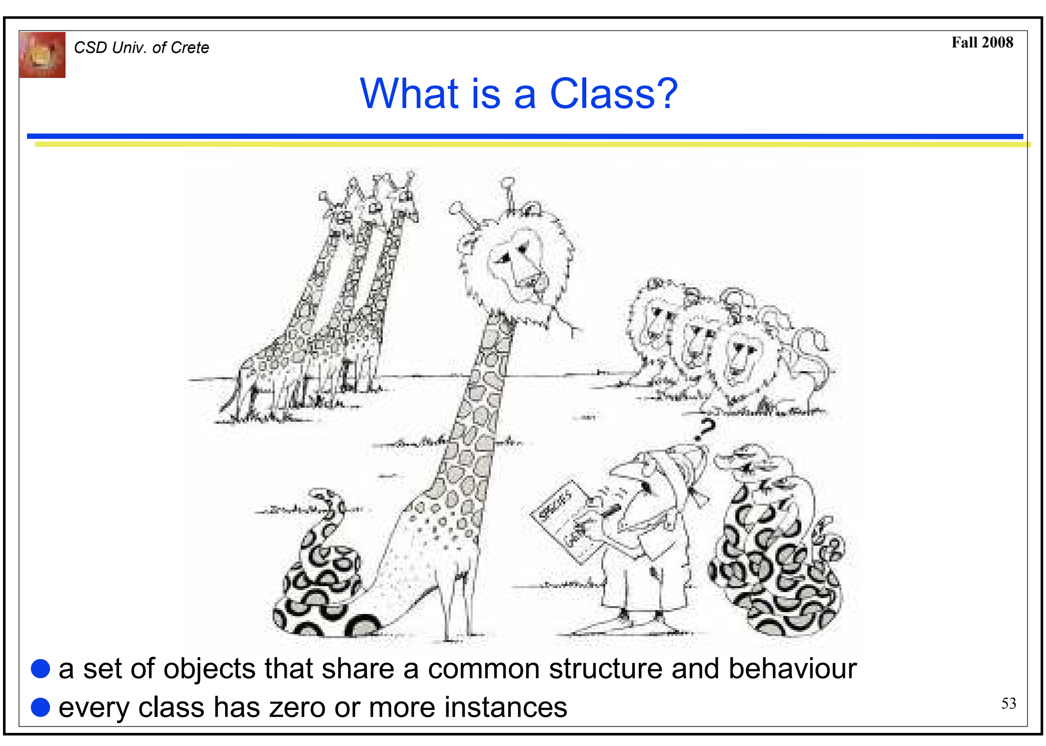

CSD Univ. of Crete
Fall 2008
A More Formal Definition
An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
An object is something that has
State
one of the possible conditions in which an object may exist
represents over time the cumulative results of its behaviour
Behavior
determines how an object acts and reacts to requests from other objects
Identity
distinguishes it from other similar objects, even if its state is identical to that of another object
STATE
BEHAVIOR
IDENTITY
48

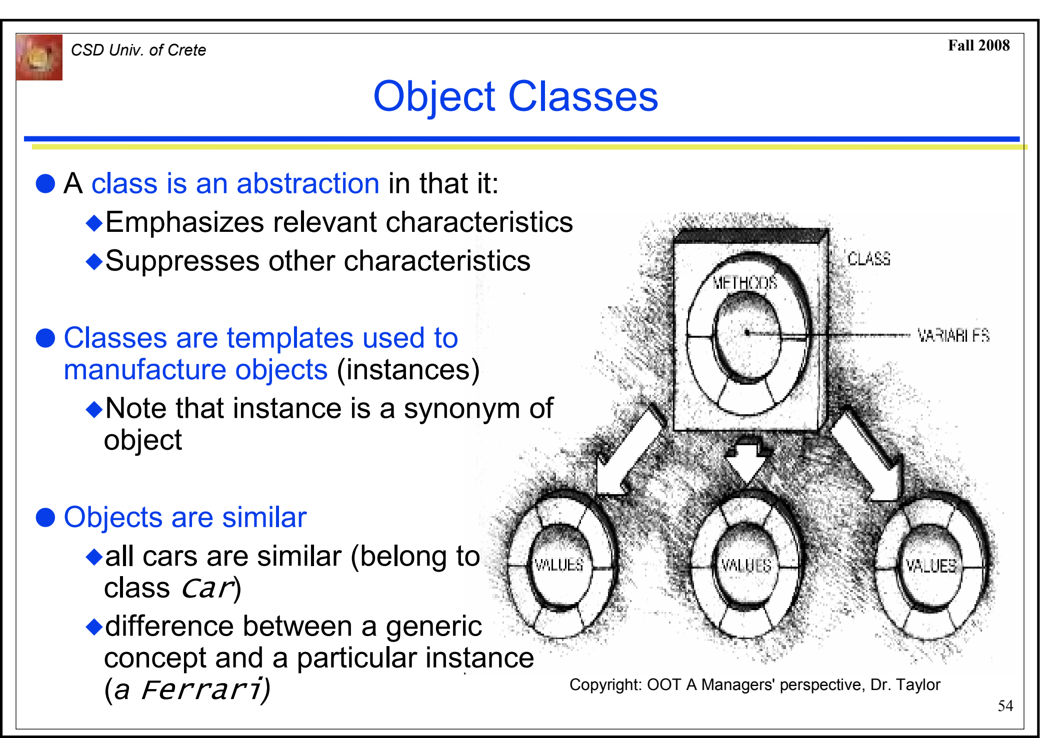

CSD Univ. of Crete
Fall 2008
What is Encapsulation?
DOG TAUNTER
WOOL TANGLER
PURRER
POUNCER
compartmentalisation of structure and behaviour so that the details of an object's implementation are hidden
49

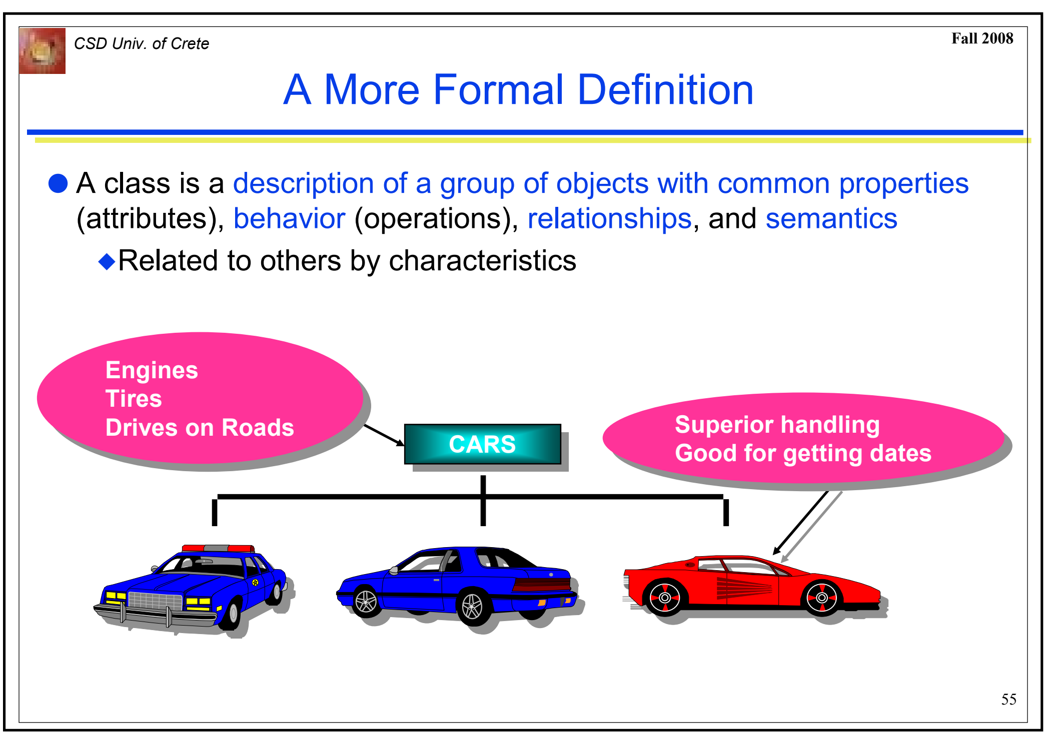

CSD Univ. of Crete
Fall 2008
OOP Key Concepts: Encapsulation
Attributes are encapsulated by the objects behavior
You don't need to know how the engine works to drive an automobile
A great deal of functionality is invisible
Turn a switch - radio comes on
Press a pedal - car accelerates
The better the design and the tighter the integration with the state of an object makes the object work better
separates implementation from interface
controlled access to data
extends the built-in types
allows for greater modularity
MOVETO
LOAD
SIZE
UNLOAD
CAPACITY
SPEED
CONTENTS
LIFT
LOWER
Copyright: OOT A Managers' perspective, Dr. Taylor
50


CSD Univ. of Crete
Fall 2008
Separation of Responsibilities
We give an object responsibility
We can provide two types of operations:
Accessors
Methods which return (state) information
Transformers
Methods which change the object (state) information
a Hire Car object
newEngine(serial)
service()
hirer
miles()
hire(cust)
avail?
clean()
51


CSD Univ. of Crete
Fall 2008
More on Encapsulation
x
y
z
op er ati on
Global variables are encapsulated in modules (now called objects)
module
module
module
OOP is a discipline that relies on objects to impose a modular structure on programs
OOP is more securely founded in an imperative language that supports the concept of encapsulation
52

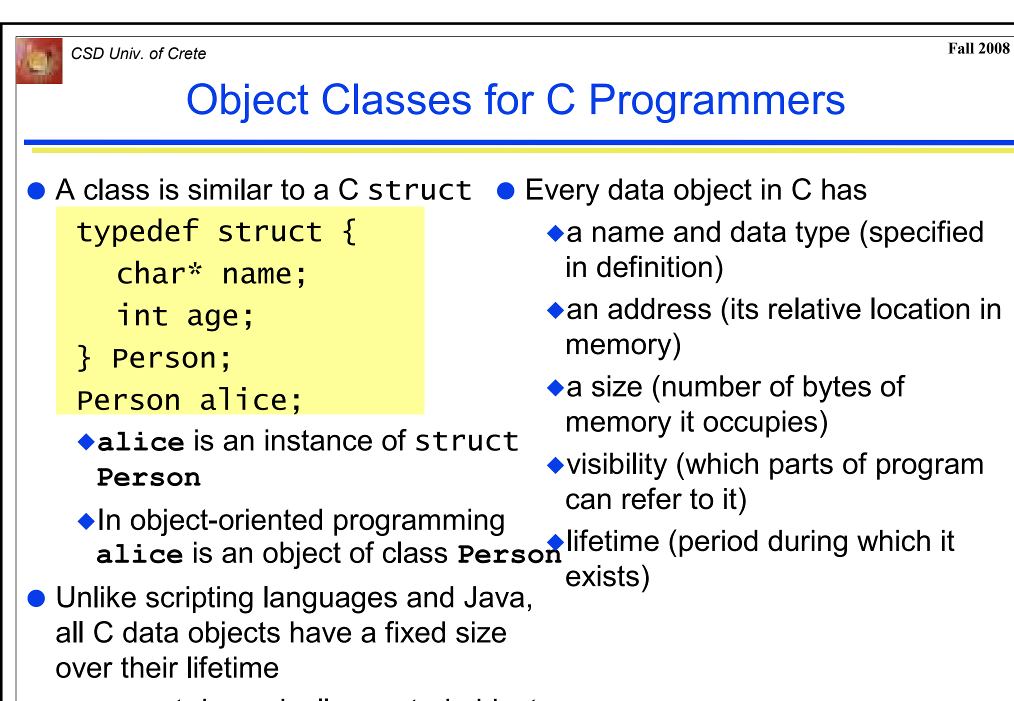

CSD Univ. of Crete
Fall 2008
What is a Class?
a set of objects that share a common structure and behaviour
every class has zero or more instances
53

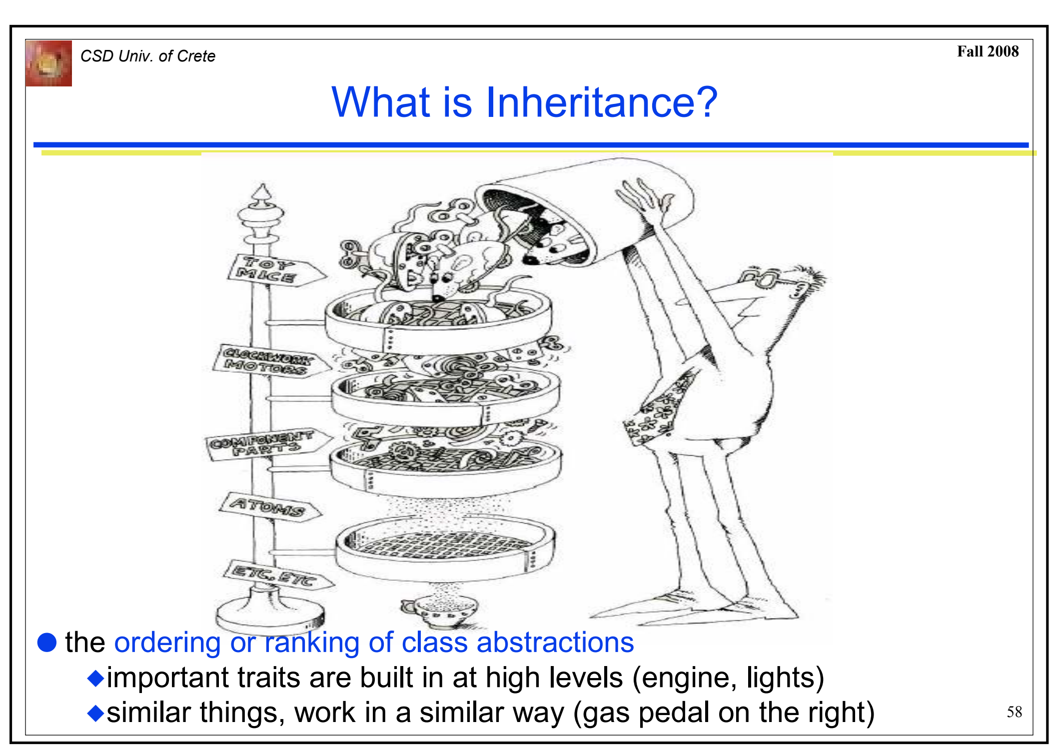

CSD Univ. of Crete
Fall 2008
Object Classes
A class is an abstraction in that it:
Emphasizes relevant characteristics
Suppresses other characteristics
Classes are templates used to manufacture objects (instances)
Note that instance is a synonym of object
Objects are similar
all cars are similar (belong to class Car)
difference between a generic concept and a particular instance (a Ferrari)
CLASS
METHODS
VARIABLES
VALUES
VALUES
VALUES
Copyright: OOT A Managers' perspective, Dr. Taylor
54

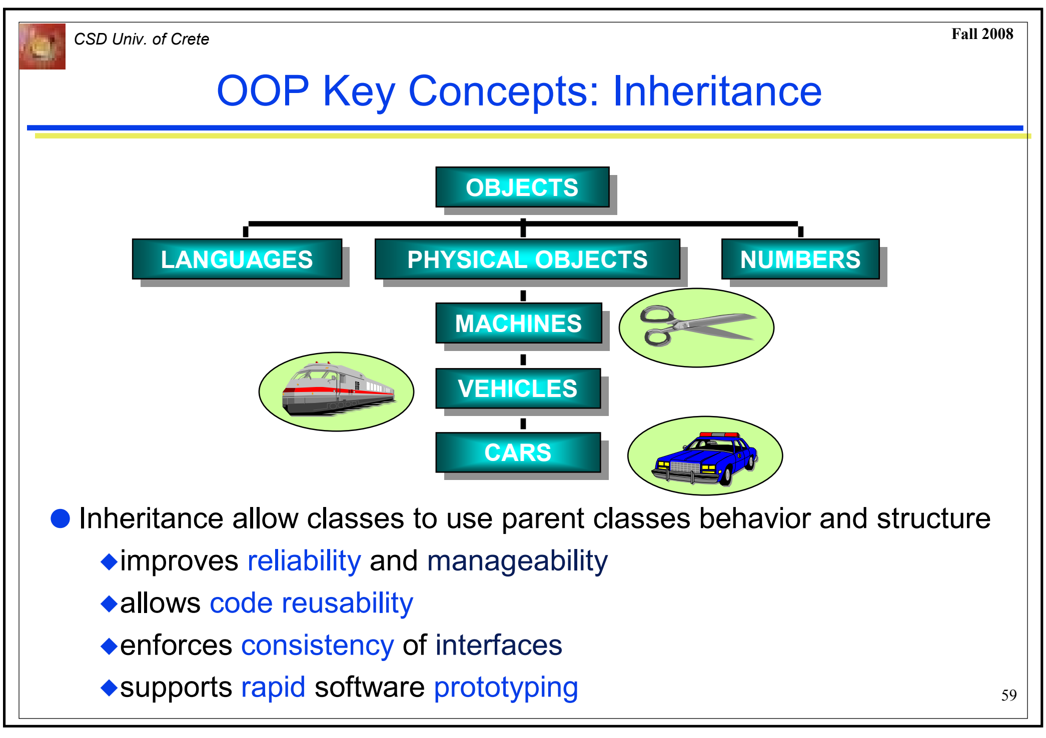


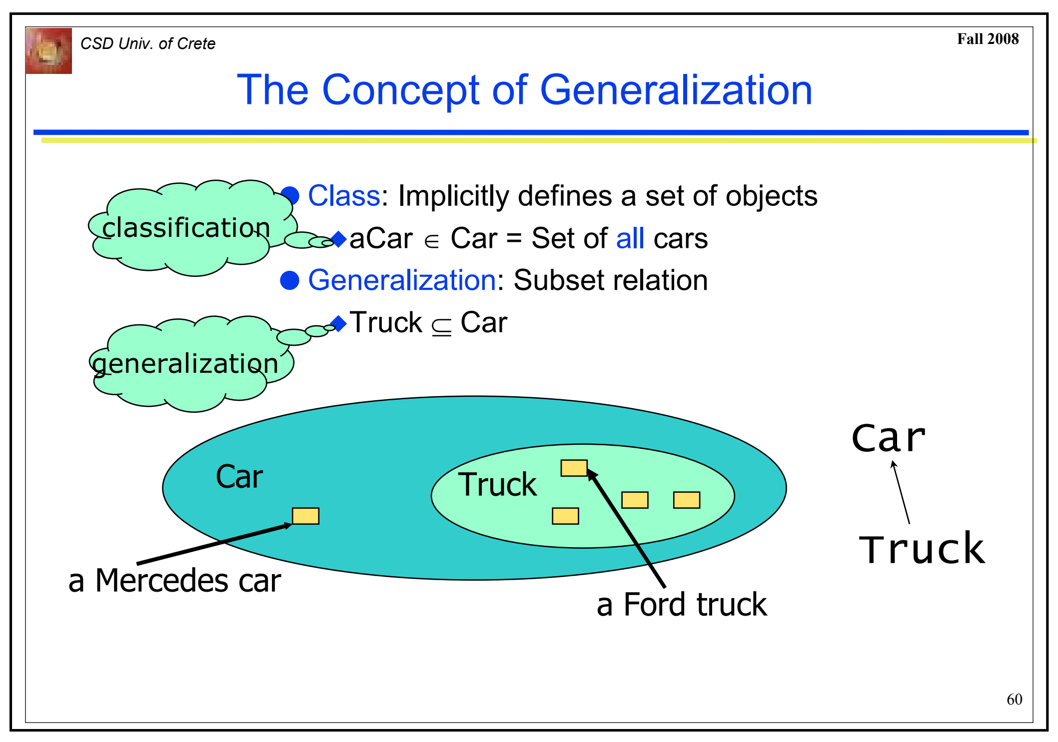
Interpretation/Representation of Objects & Classes

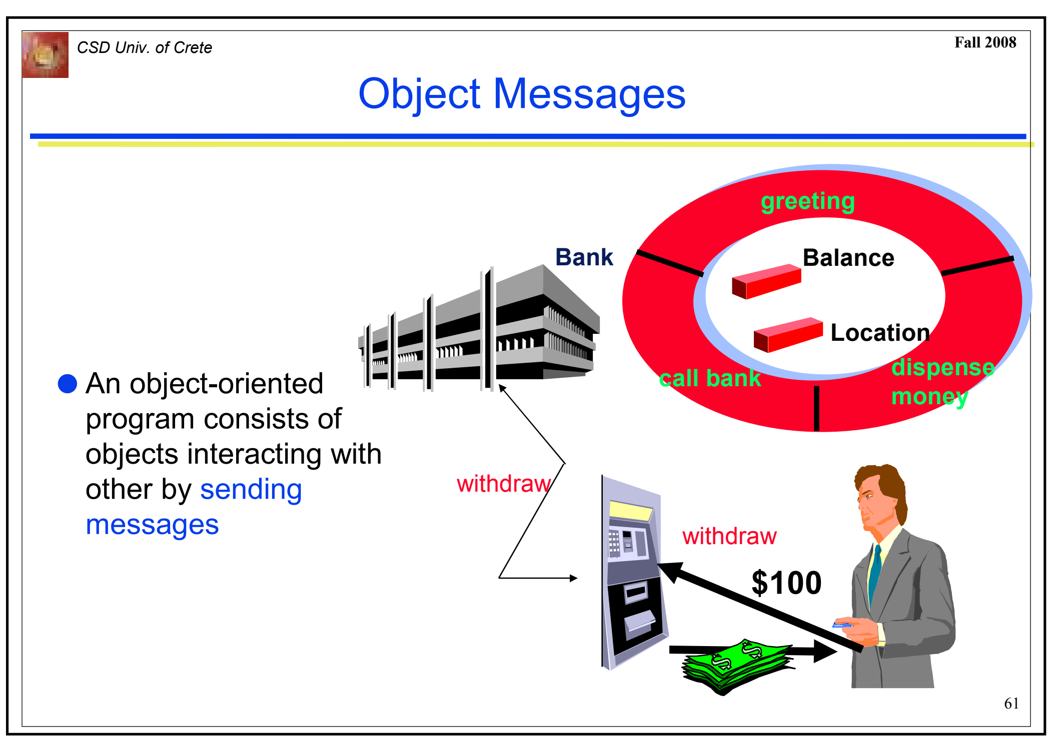
	Interpretation in the real world	Representation in the computer program
Object	An object represents anything in the real world that can be distinctly identified	An object has a unique identity, a state, and behaviors
Class	A class represents a set of objects with similar characteristics and behaviors. These objects are called instance of the class	A class characterizes the structure of states and behaviors that are shared by all its instances

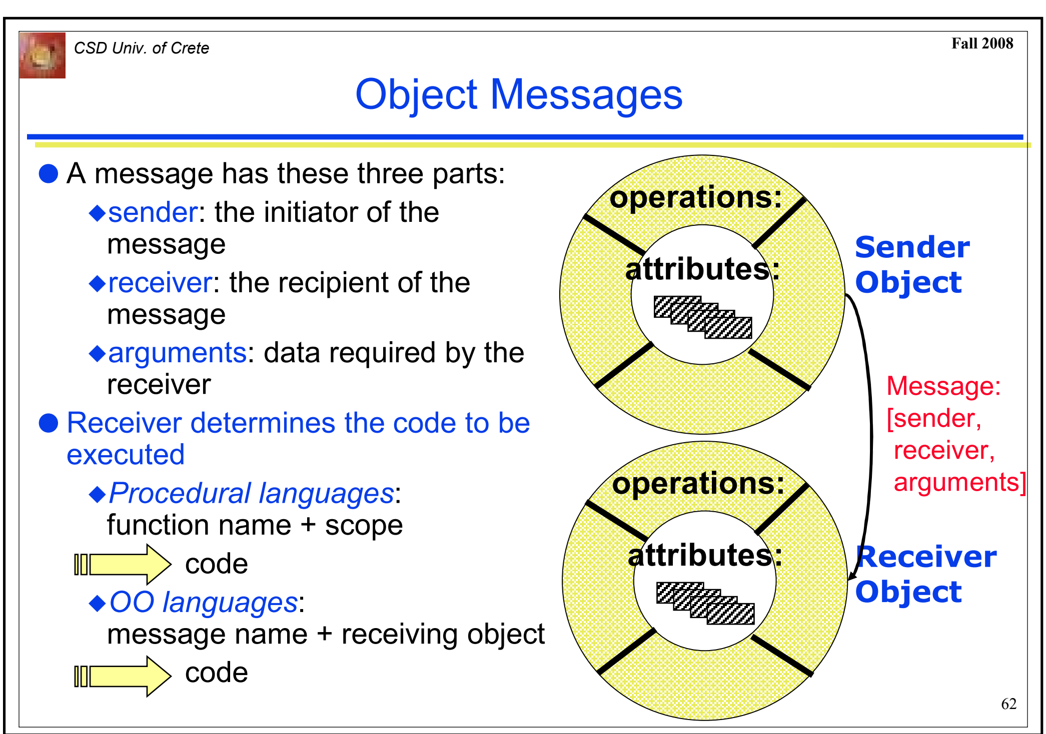

CSD Univ. of Crete
Fall 2008
Object Classes for C Programmers
A class is similar to a C struct
typedef struct {
char* name;
int age;
} Person;
Person alice;
alice is an instance of struct Person
In object-oriented programming alice is an object of class Person
Unlike scripting languages and Java, all C data objects have a fixed size over their lifetime
except dynamically created objects
Every data object in C has
a name and data type (specified in definition)
an address (its relative location in memory)
a size (number of bytes of memory it occupies)
visibility (which parts of program can refer to it)
lifetime (period during which it exists)
57

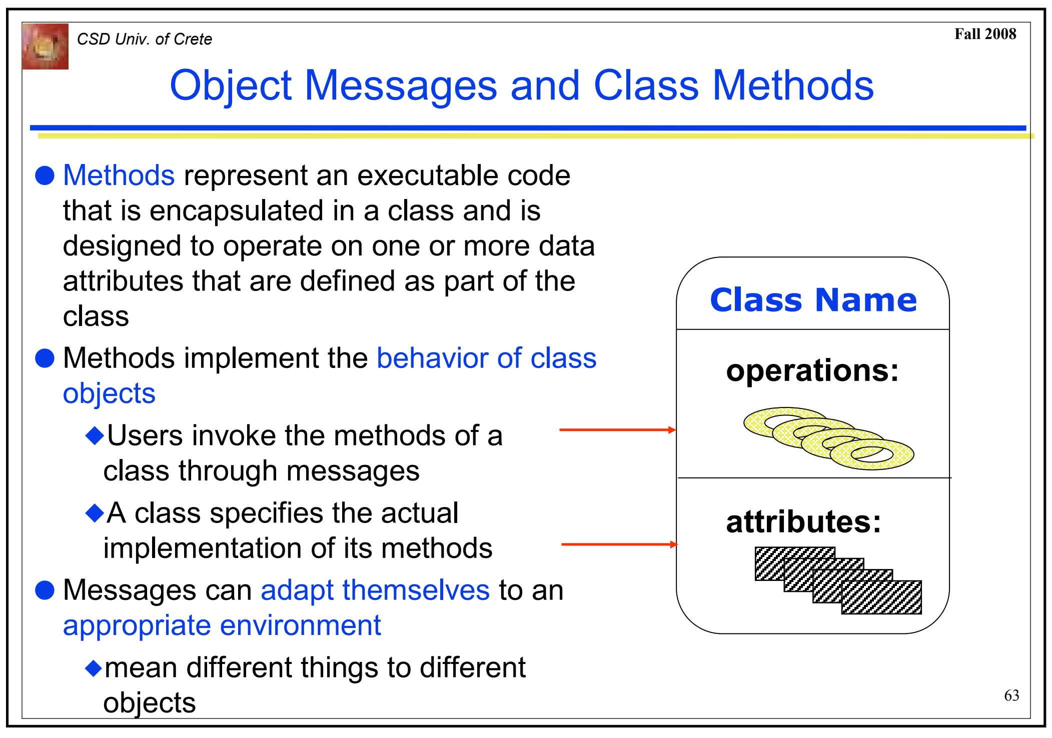

CSD Univ. of Crete
Fall 2008
What is Inheritance?
TOY MICE
CLOCKWORK MOTORS
COMPONENT PARTS
ATOMS
ETC. ETC
the ordering or ranking of class abstractions
important traits are built in at high levels (engine, lights)
similar things, work in a similar way (gas pedal on the right)
58

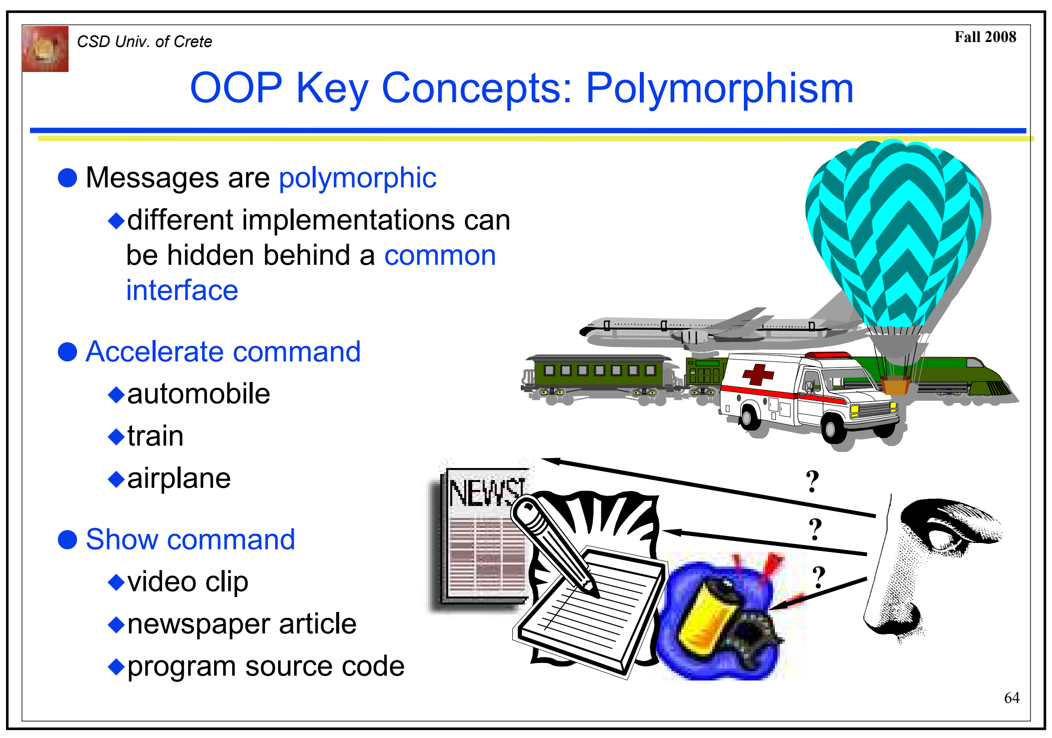

CSD Univ. of Crete
Fall 2008
OOP Key Concepts: Inheritance
OBJECTS
LANGUAGES
PHYSICAL OBJECTS
NUMBERS
MACHINES
VEHICLES
CARS
Inheritance allow classes to use parent classes behavior and structure
improves reliability and manageability
allows code reusability
enforces consistency of interfaces
supports rapid software prototyping
59

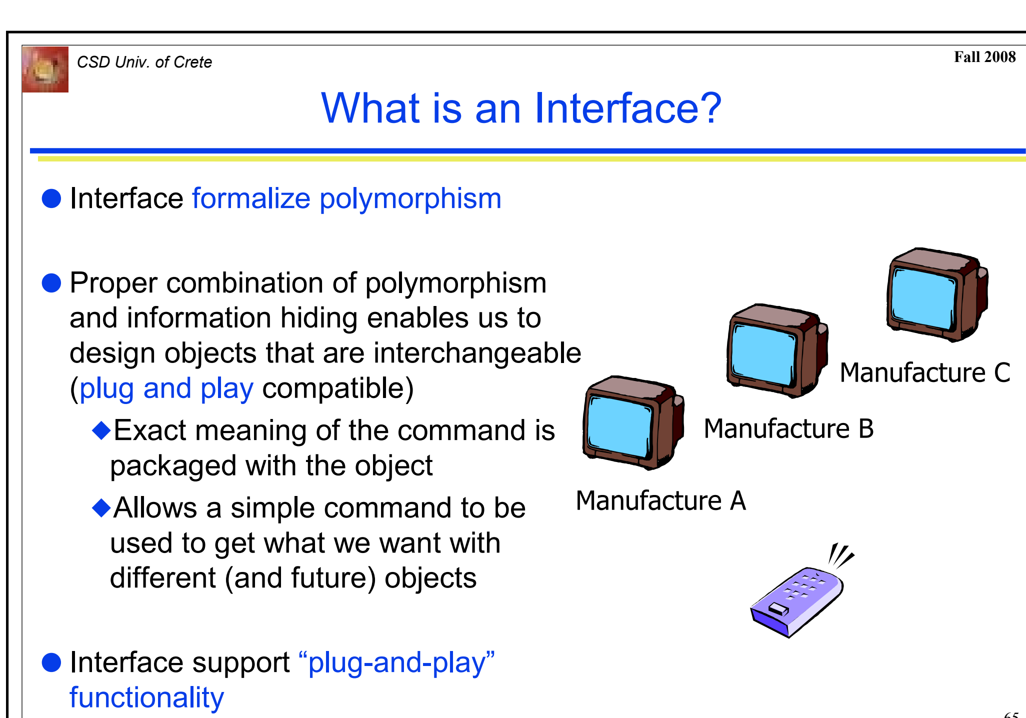

CSD Univ. of Crete
Fall 2008
The Concept of Generalization
classification
Class: Implicitly defines a set of objects
aCar ∈ Car = Set of all cars
Generalization: Subset relation
generalization
Truck ⊆ Car
Car
Truck
a Mercedes car
a Ford truck
Car
Truck
60

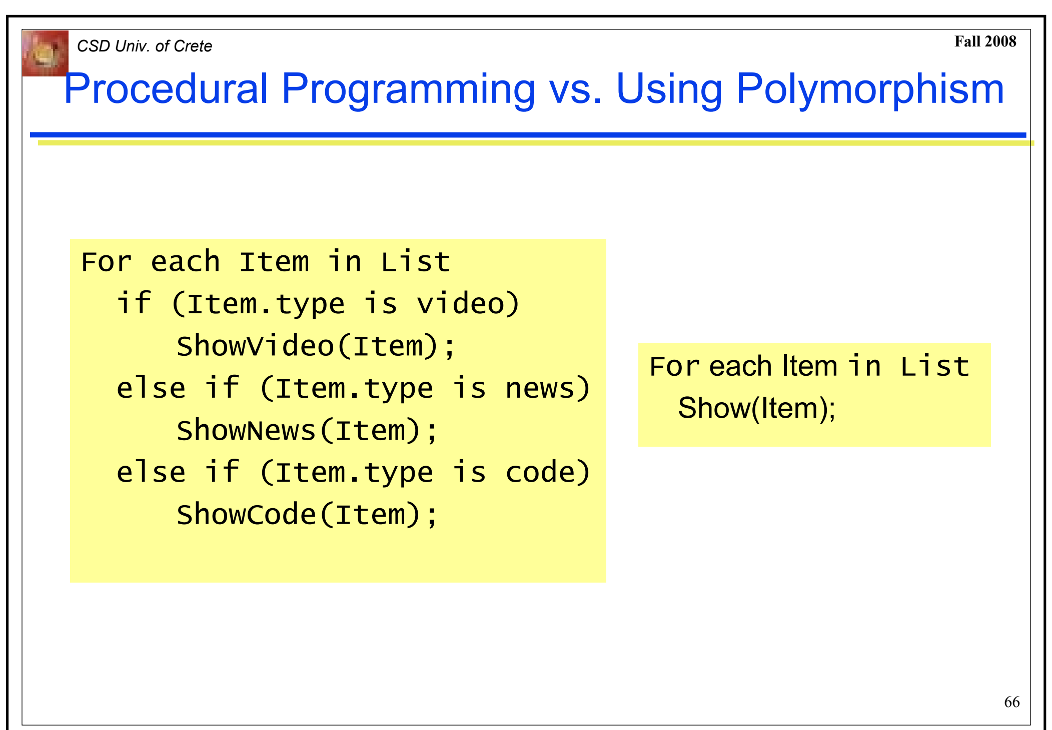

CSD Univ. of Crete
Fall 2008
Object Messages
An object-oriented program consists of objects interacting with other by sending messages
Bank
greeting
Balance
Location
call bank
dispense money
withdraw
withdraw
$100
61

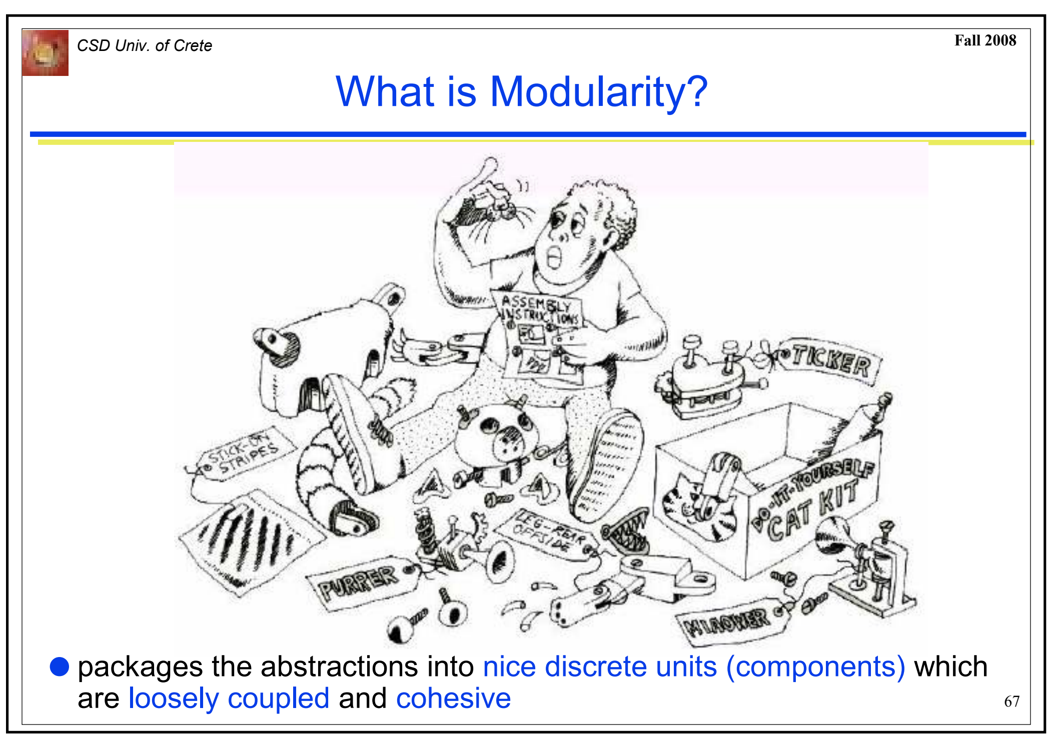

CSD Univ. of Crete
Fall 2008
Object Messages
A message has these three parts:
sender: the initiator of the message
receiver: the recipient of the message
arguments: data required by the receiver
Receiver determines the code to be executed
Procedural languages: function name + scope
code
OO languages: message name + receiving object
code
operations:
attributes:
Sender Object
Message: [sender, receiver, arguments]
operations:
attributes:
Receiver Object
62

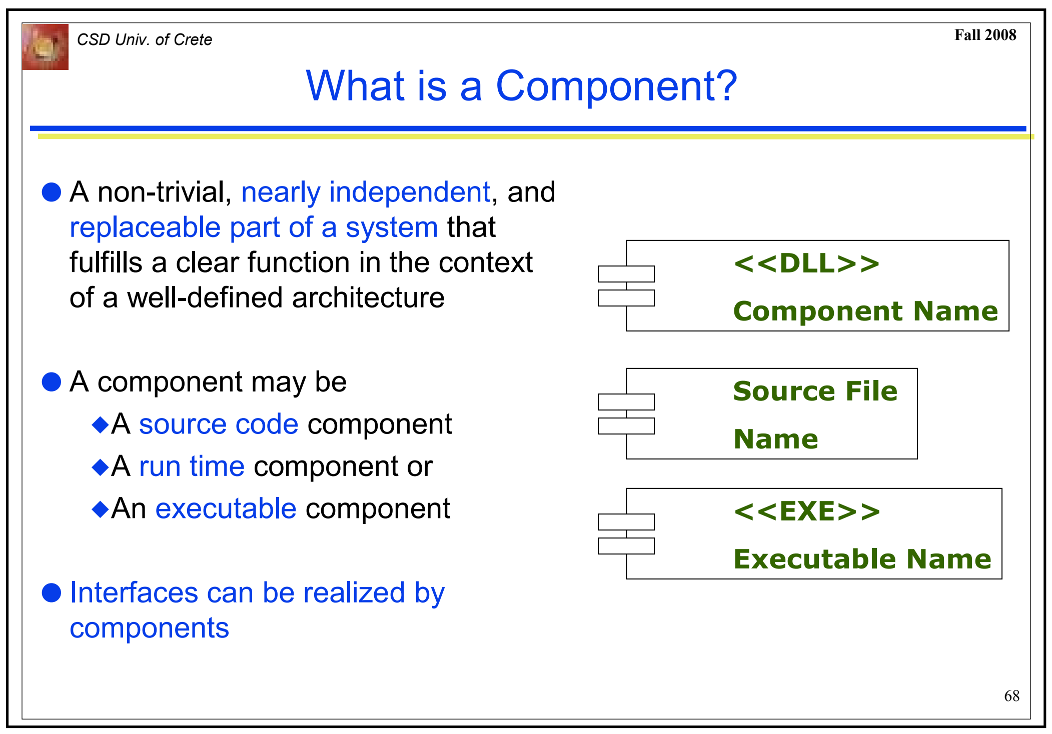

CSD Univ. of Crete
Fall 2008
Object Messages and Class Methods
Methods represent an executable code that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class
Methods implement the behavior of class objects
Users invoke the methods of a class through messages
A class specifies the actual implementation of its methods
Messages can adapt themselves to an appropriate environment
mean different things to different objects
Class Name
operations:
attributes:
63

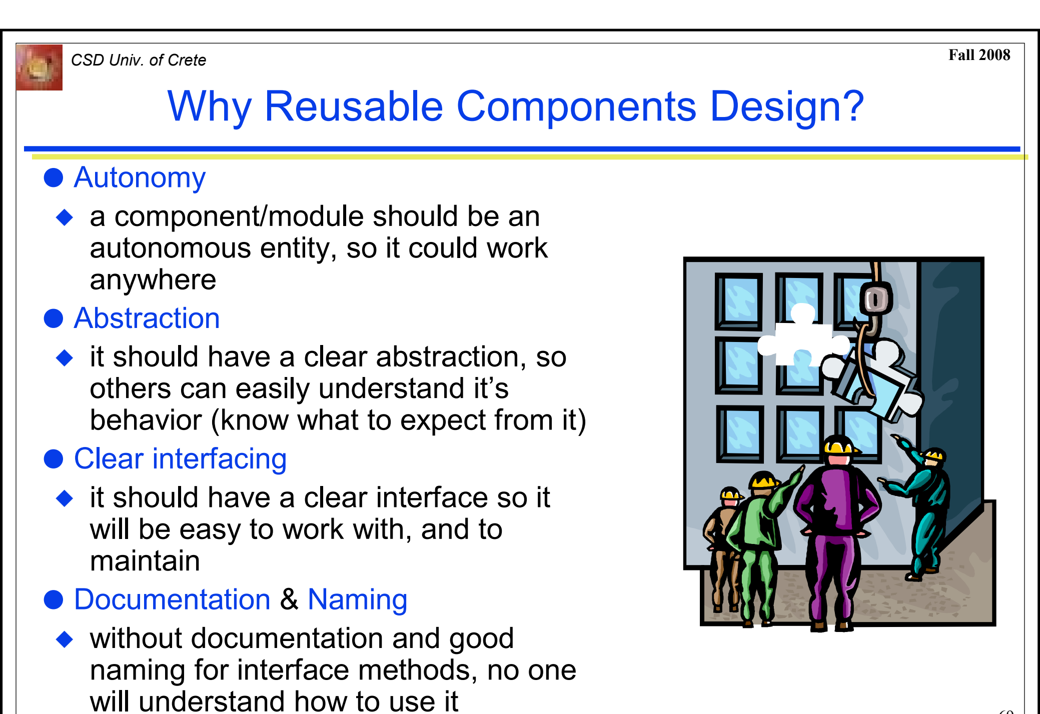

CSD Univ. of Crete
Fall 2008
OOP Key Concepts: Polymorphism
Messages are polymorphic
different implementations can be hidden behind a common interface
Accelerate command
automobile
train
airplane
Show command
video clip
newspaper article
program source code
NEWS
?
?
?
64

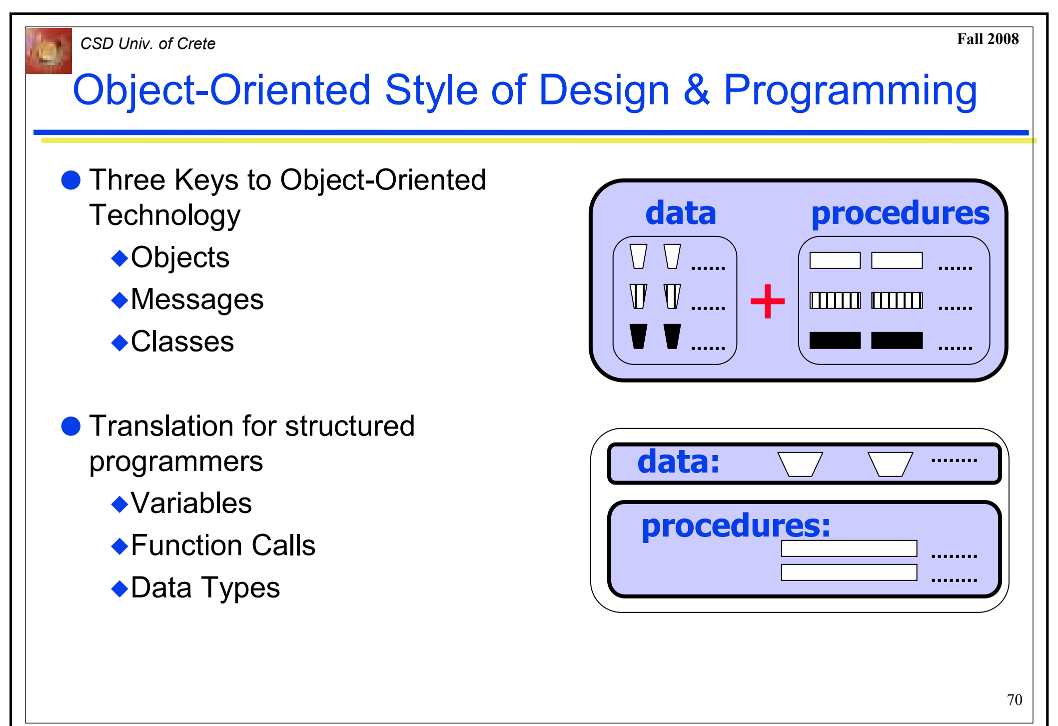

CSD Univ. of Crete
Fall 2008
What is an Interface?
Interface formalize polymorphism
Proper combination of polymorphism and information hiding enables us to design objects that are interchangeable (plug and play compatible)
Exact meaning of the command is packaged with the object
Allows a simple command to be used to get what we want with different (and future) objects
Manufacture C
Manufacture B
Manufacture A
Interface support "plug-and-play" functionality
65

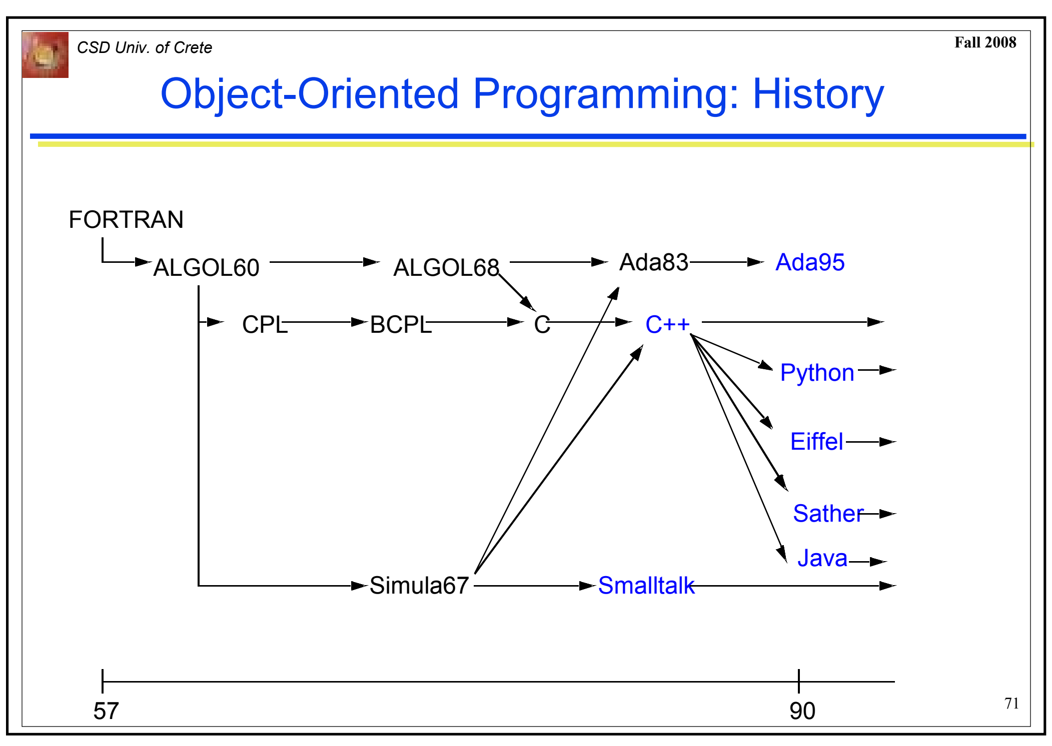

CSD Univ. of Crete
Fall 2008
Procedural Programming vs. Using Polymorphism
For each Item in List
if (Item.type is video)
ShowVideo(Item);
else if (Item.type is news)
ShowNews(Item);
else if (Item.type is code)
ShowCode(Item);
For each Item in List
Show(Item);
66

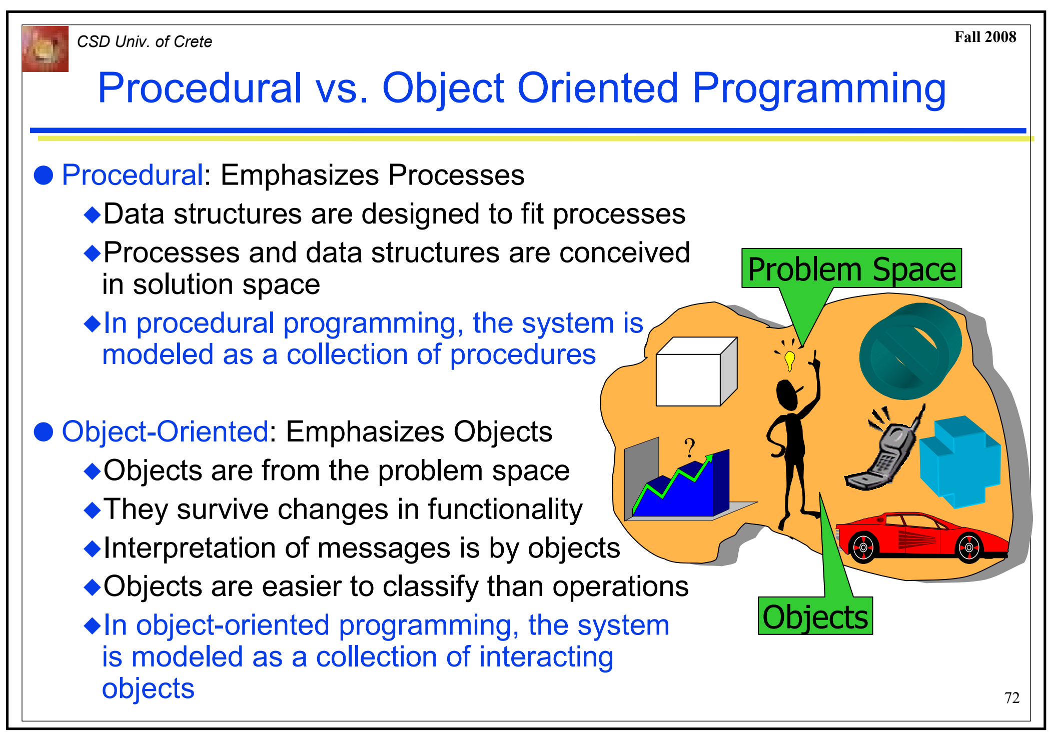

CSD Univ. of Crete
Fall 2008
What is Modularity?
ASSEMBLY INSTRUCTIONS
TICKER
STICK-ON STRIPES
DO-IT-YOURSELF CAT KIT
LEG-REAR OFFSIDE
PURRER
MIAOWER
packages the abstractions into nice discrete units (components) which are loosely coupled and cohesive
67

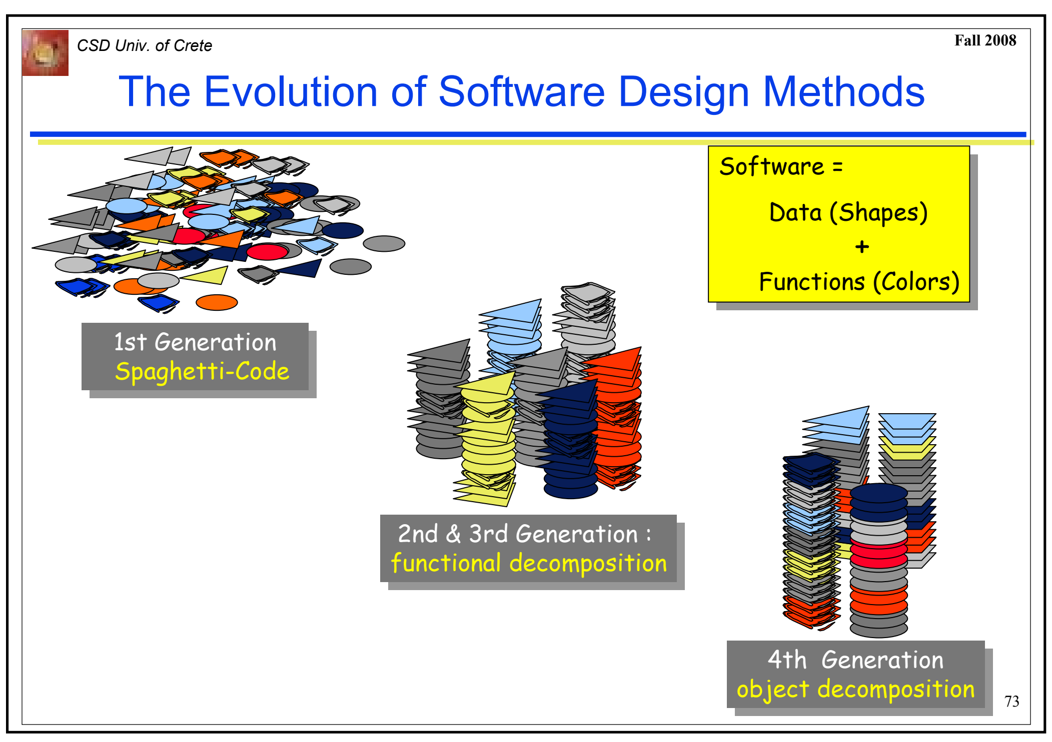

CSD Univ. of Crete
Fall 2008
What is a Component?
A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
A component may be
A source code component
A run time component or
An executable component
Interfaces can be realized by components
<<DLL>>
Component Name
Source File
Name
<<EXE>>
Executable Name
68

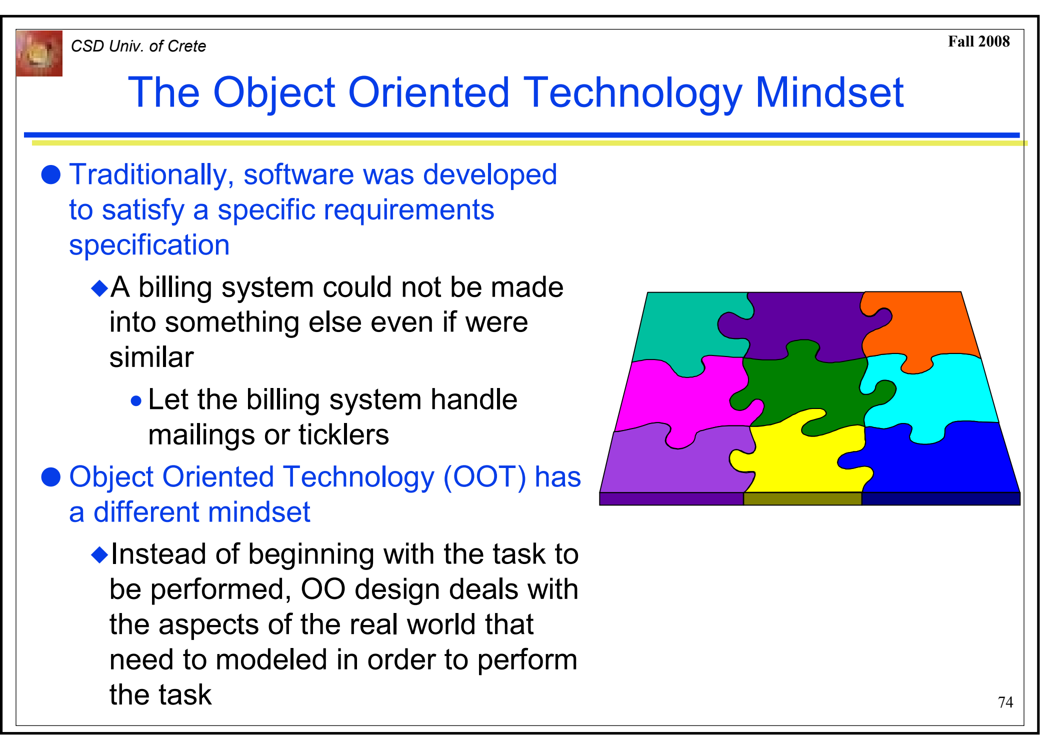

CSD Univ. of Crete
Fall 2008
Why Reusable Components Design?
Autonomy
a component/module should be an autonomous entity, so it could work anywhere
Abstraction
it should have a clear abstraction, so others can easily understand it's behavior (know what to expect from it)
Clear interfacing
it should have a clear interface so it will be easy to work with, and to maintain
Documentation & Naming
without documentation and good naming for interface methods, no one will understand how to use it
69


CSD Univ. of Crete
Fall 2008
Object-Oriented Style of Design & Programming
Three Keys to Object-Oriented Technology
Objects
Messages
Classes
Translation for structured programmers
Variables
Function Calls
Data Types
data
procedures
+
data:
procedures:
70


CSD Univ. of Crete
Fall 2008
Object-Oriented Programming: History
FORTRAN
ALGOL60
ALGOL68
Ada83
Ada95
CPL
BCPL
C
C++
Python
Eiffel
Sather
Java
Simula67
Smalltalk
57
90
71

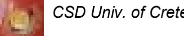

CSD Univ. of Crete
Fall 2008
Procedural vs. Object Oriented Programming
Procedural: Emphasizes Processes
Data structures are designed to fit processes
Processes and data structures are conceived in solution space
In procedural programming, the system is modeled as a collection of procedures
Object-Oriented: Emphasizes Objects
Objects are from the problem space
They survive changes in functionality
Interpretation of messages is by objects
Objects are easier to classify than operations
In object-oriented programming, the system is modeled as a collection of interacting objects
Problem Space
Objects
72


CSD Univ. of Crete
Fall 2008
The Evolution of Software Design Methods
Software =
Data (Shapes)
+
Functions (Colors)
1st Generation
Spaghetti-Code
2nd & 3rd Generation :
functional decomposition
4th Generation
object decomposition
73


CSD Univ. of Crete
Fall 2008
The Object Oriented Technology Mindset
Traditionally, software was developed to satisfy a specific requirements specification
A billing system could not be made into something else even if were similar
Let the billing system handle mailings or ticklers
Object Oriented Technology (OOT) has a different mindset
Instead of beginning with the task to be performed, OO design deals with the aspects of the real world that need to modeled in order to perform the task
74

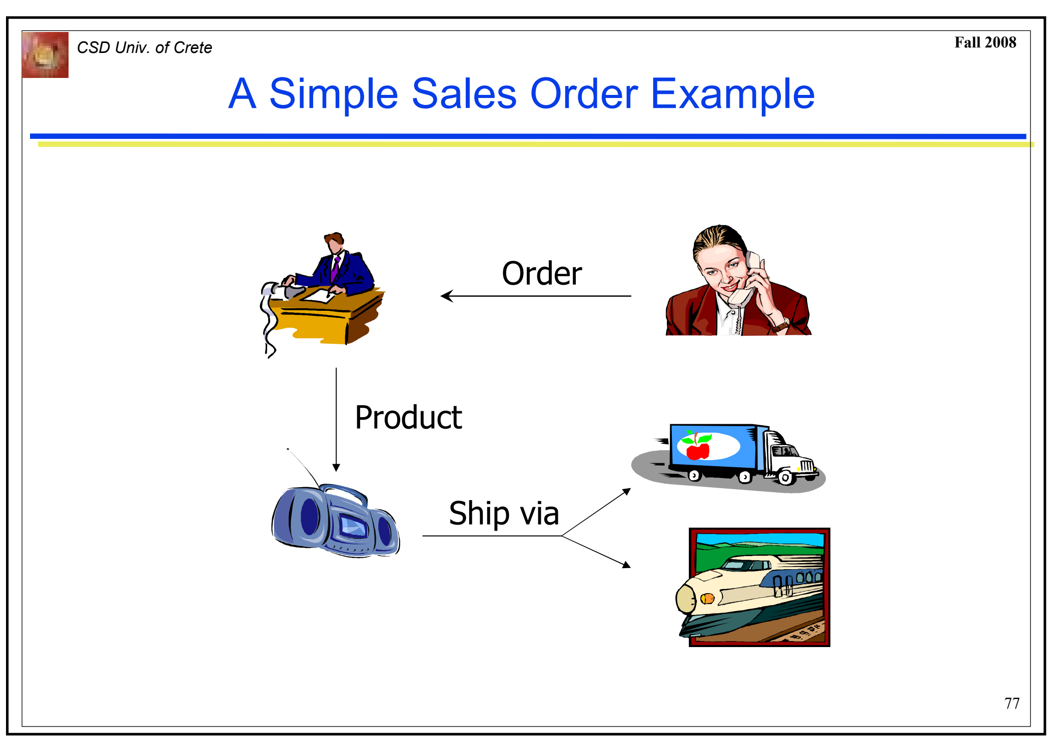
A Wish for Reuse

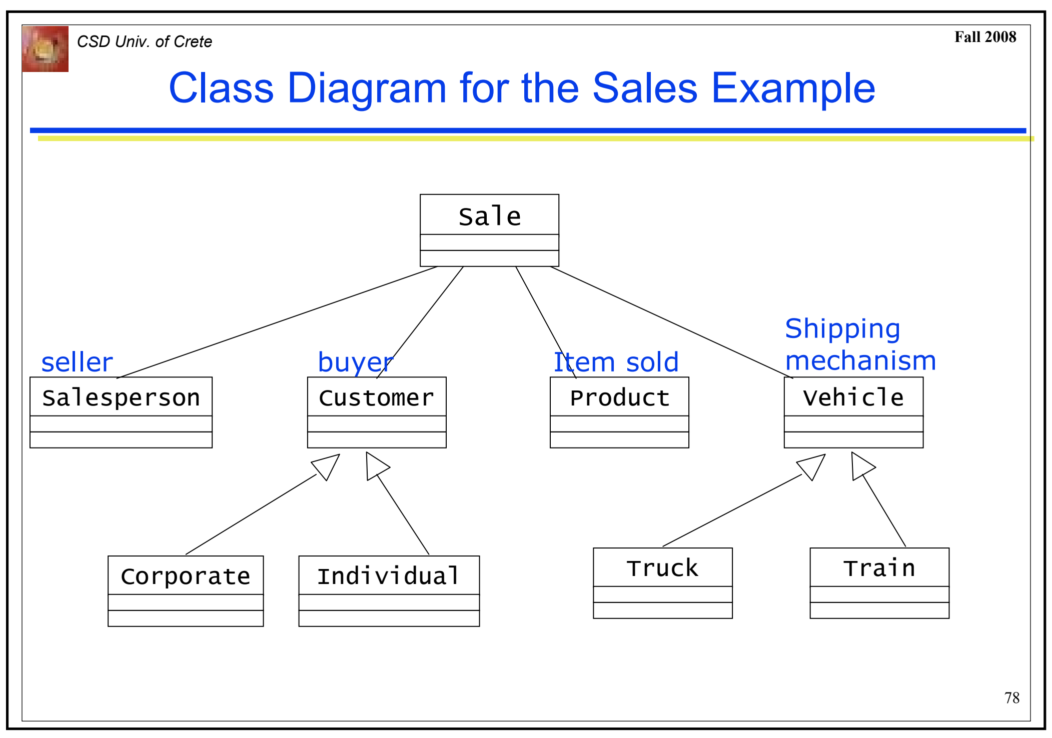
- Traditional software started from scratch
  - easier than converting old code--specific task
- Object Oriented Technology stresses reuse
  - objects are the building blocks
  - majority of time spent assembling proven components: e.g,. Graphical User Interface (GUI)
    - Borland's OWL, MS's MFC, or Java Swing
  - But reuse is hard to obtain!
    - Extreme programmers don't strive for it, they just do what they are getting paid to do

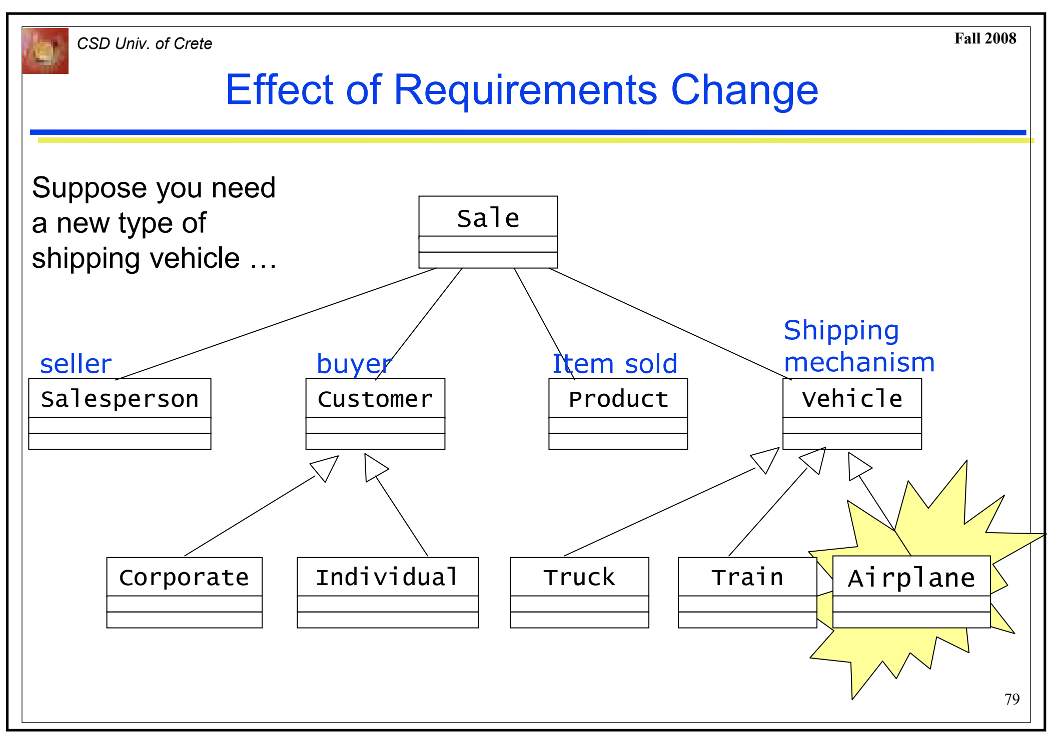


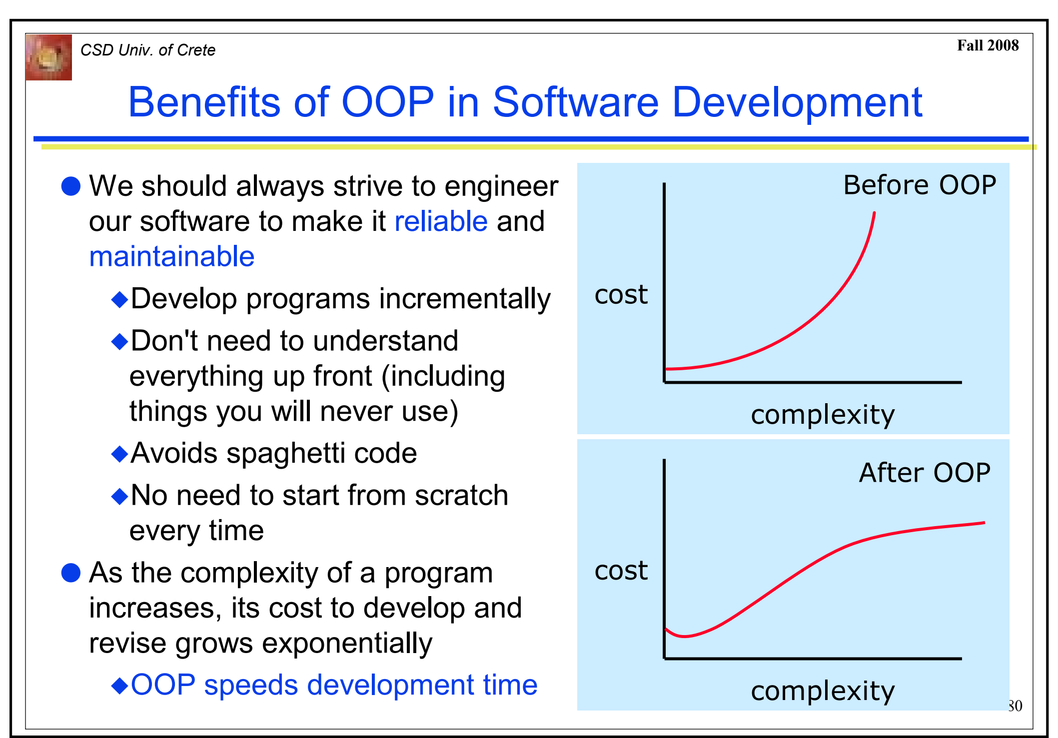
The Promise of the Approach

- Object Oriented Technology offers
  - techniques for creating flexible, natural software modules
  - systems that are much easier to adapt to new demands
  - reuse shortens the development life cycle
  - systems are more understandable and maintainable
    - easier to remember 50 real world classes rather than 500 functions!
- Basic corporate operations change more slowly than the information needs
  - software based on corporate models have a longer life span
- Do you believe it has been easy for corporations to switch to this new technology?


CSD Univ. of Crete
Fall 2008
A Simple Sales Order Example
Order
Product
Ship via
77


CSD Univ. of Crete
Fall 2008
Class Diagram for the Sales Example
Sale
seller
buyer
Item sold
Shipping mechanism
Salesperson
Customer
Product
Vehicle
Corporate
Individual
Truck
Train
78


CSD Univ. of Crete
Fall 2008
Effect of Requirements Change
Suppose you need a new type of shipping vehicle …
Sale
seller
Salesperson
buyer
Customer
Item sold
Product
Shipping mechanism
Vehicle
Corporate
Individual
Truck
Train
Airplane
79


CSD Univ. of Crete
Fall 2008
Benefits of OOP in Software Development
We should always strive to engineer our software to make it reliable and maintainable
Develop programs incrementally
Don't need to understand everything up front (including things you will never use)
Avoids spaghetti code
No need to start from scratch every time
As the complexity of a program increases, its cost to develop and revise grows exponentially
OOP speeds development time
Before OOP
cost
complexity
After OOP
cost
complexity