**Design Patterns**

* organization, purpose and context of each.
* Give a specific example of using each design pattern.
* Give appropriate UML describing each design pattern
* and, for those design patterns which appear as part of the CN1 language definition, be able to explain the relevant CN1 interfaces and/or classes and how they are used.

**Creational**

Factory Method

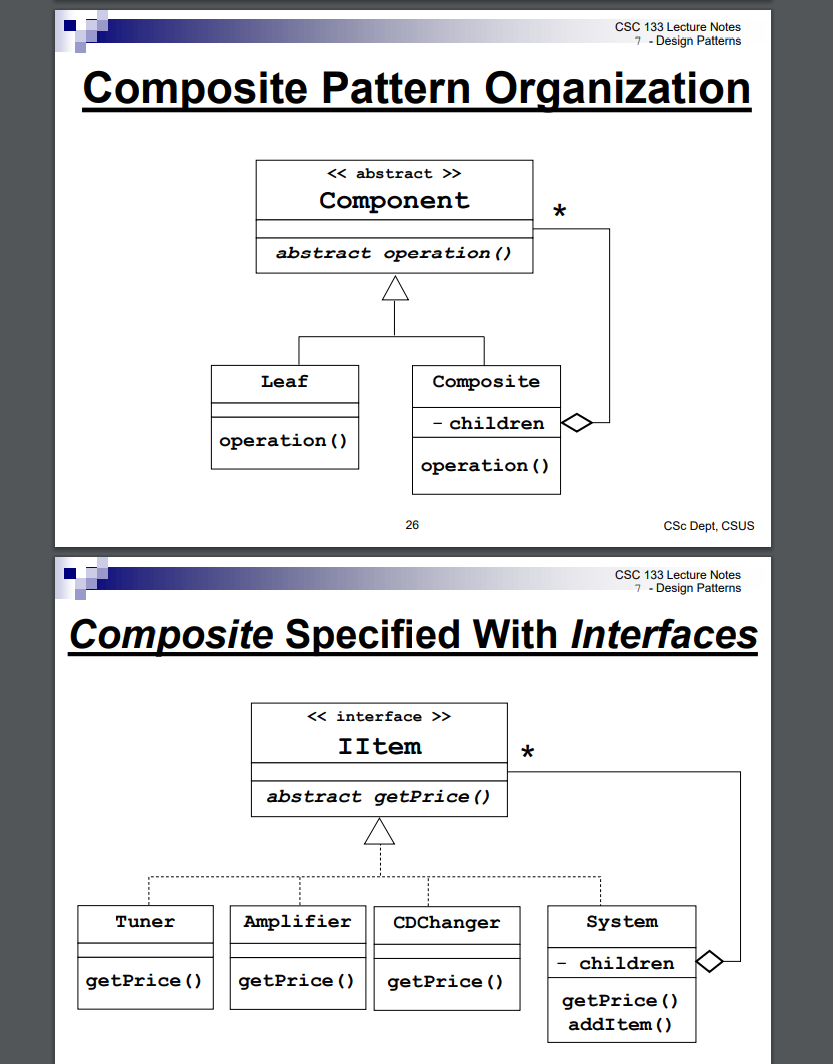
* Sometimes a class can’t anticipate the class of objects it must create
* It is sometimes better to delegate specification of object types to subclasses
* It is frequently desirable to avoid binding application-specific classes into a set of code
* EXAMPLE MAZE GAME
* //factory methods - each returns a MazeComponent of a given type
* public Maze makeMaze() { return new Maze() ; } public Room makeRoom(int id) { return new Room(id) ; }

Singleton

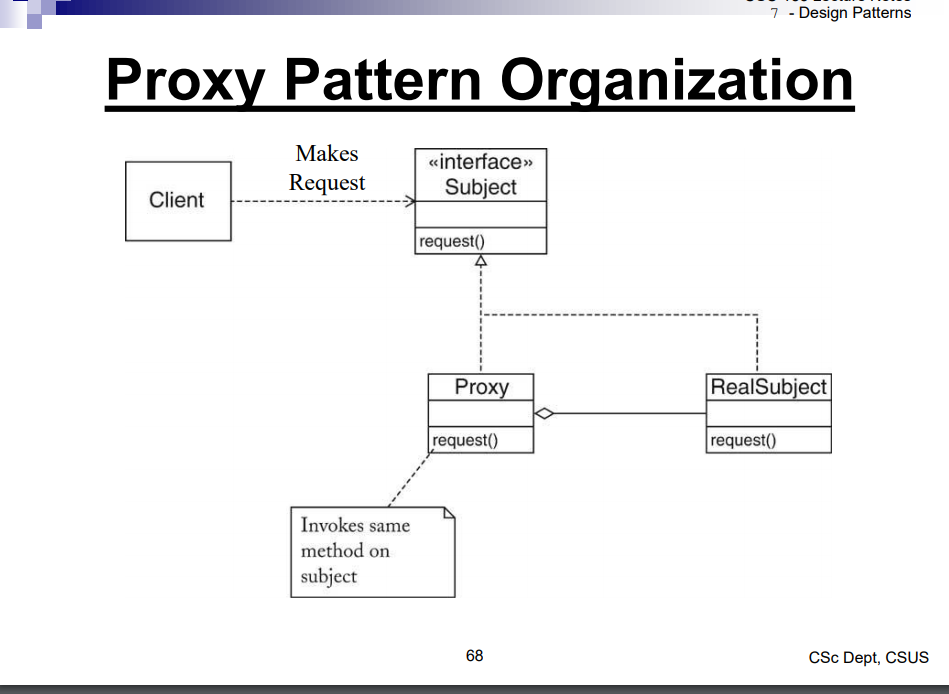
* The singleton pattern is a software design pattern that restricts the instantiation of a class to one object.
* This pattern involves a single class which is responsible to create an object while making sure that only single object gets created.
* EXAMPLE – Print Spooler, Audio Player

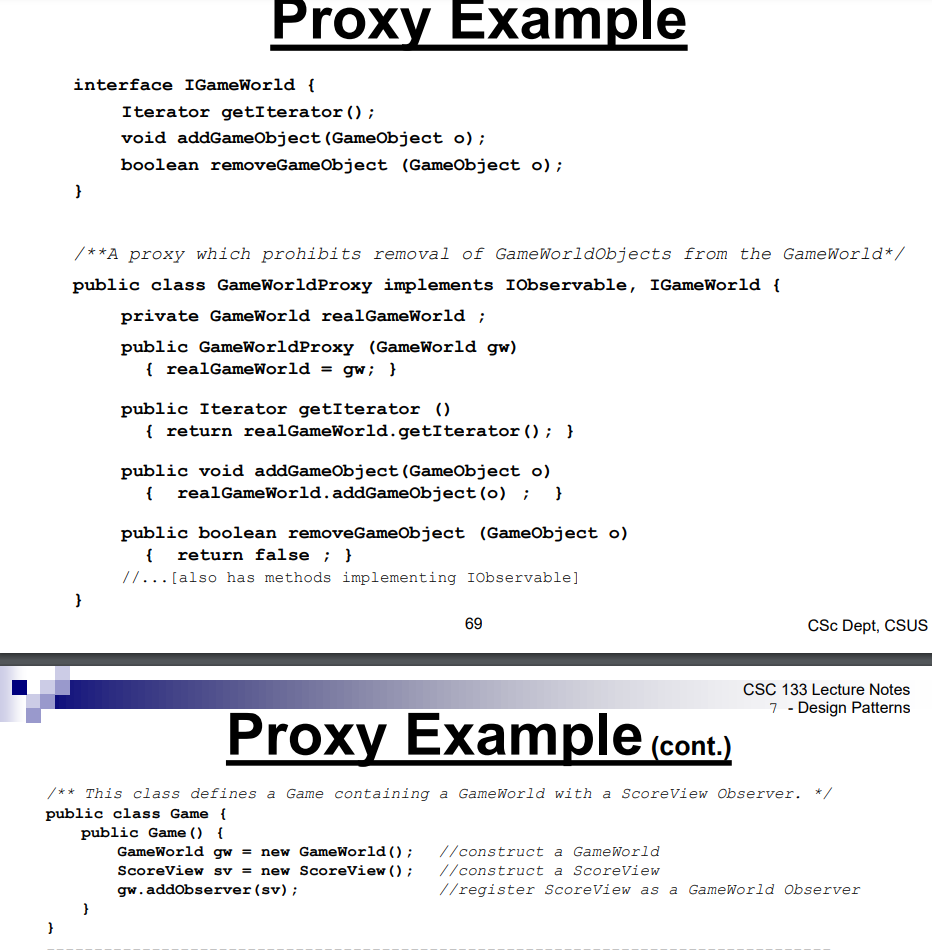
**Structural**

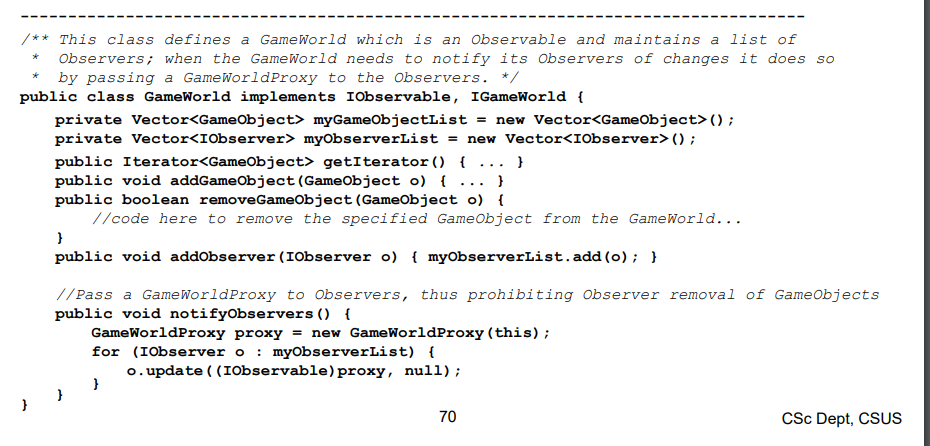
Composite

* Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy.
* EXAMPLE – store that sells stereo components (tuners, amplifiers, cdchangers…etc)
* 

Proxy

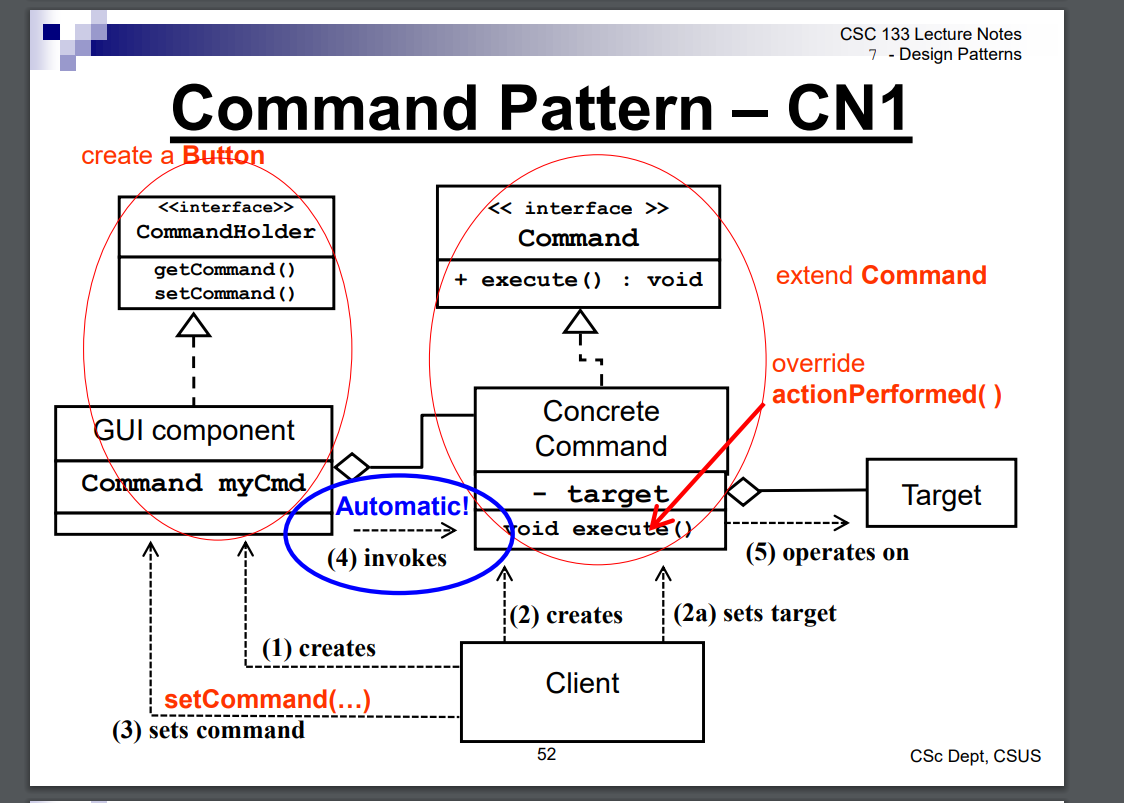
* Protection Proxy – controls access
* Virtual Proxy – acts as a stand-in
* Remote Proxy – local stand-in for object in another address space
* 



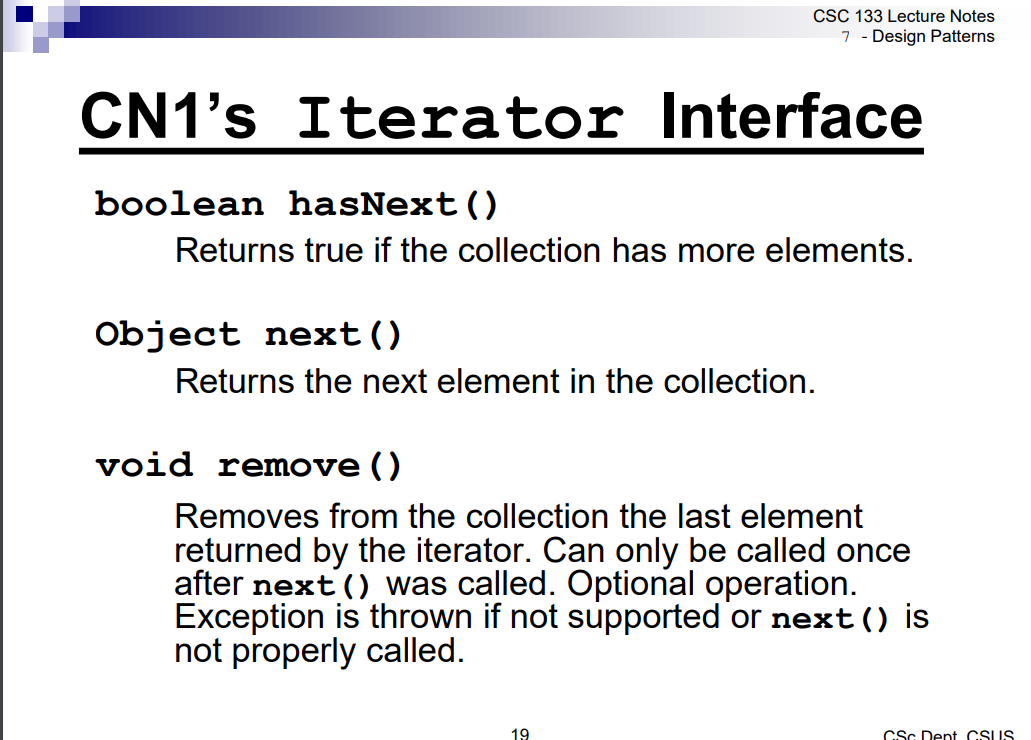
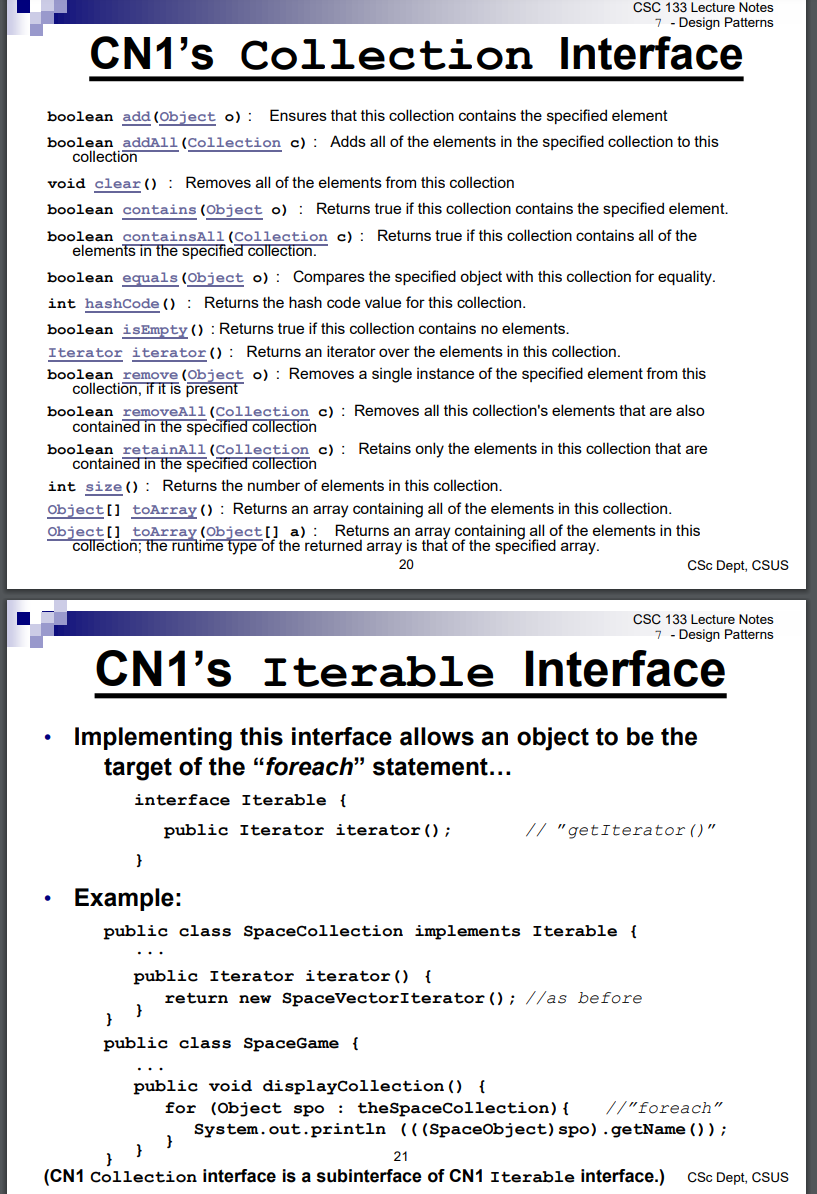


**Behavioral**

Command

* Need to avoid having multiple copies of the code that performs the same operation invoked from different sources
* Desire to separate code implementing a command from the object which invokes it
* 

Iterator

* This pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.
* EXAMPLE
  + GameWorld has a set of game characters
  + Screen view needs to display the characters
  + Screen view does not need know the data structure
* 
* 

Observer

* An object stores data that changes regularly
* Various clients use the data in different ways
* Clients need to know when the data changes
* Code that is associated with the object that stores data should not need to change when new clients are added

Strategy

* A variety of algorithms exists to perform a particular operation
* The client needs to be able to select/change the choice of algorithm at run-time.