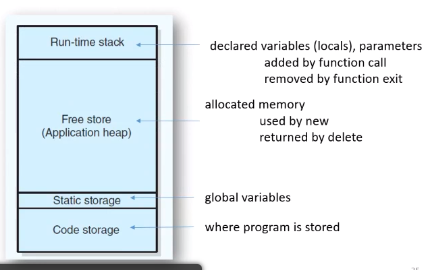
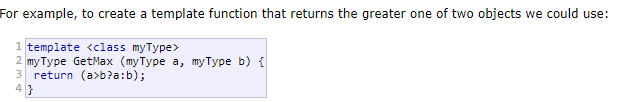
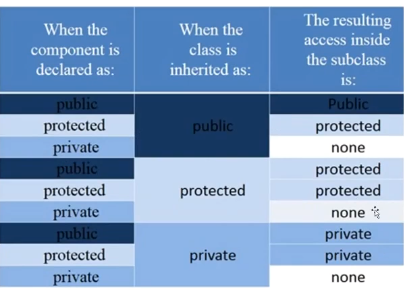
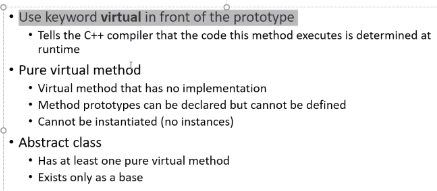
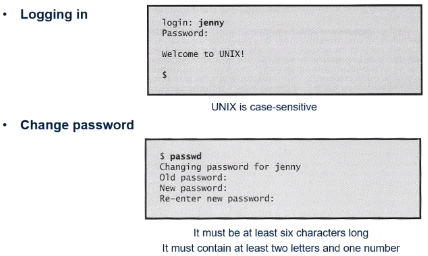
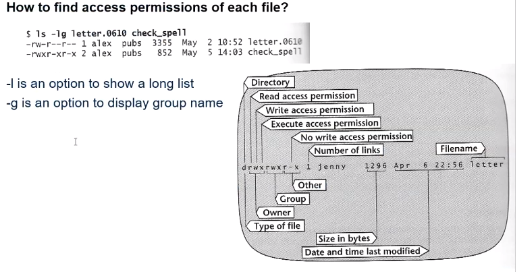
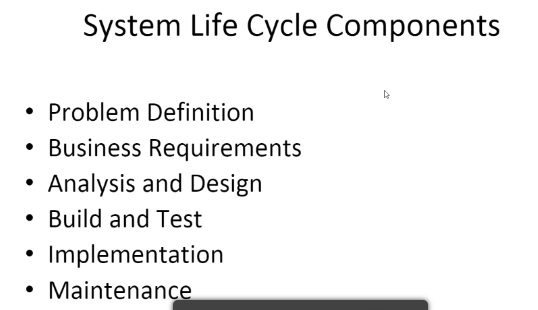
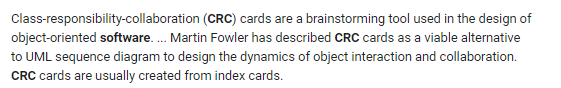
* Four principles of object-oriented programming
  + Encapsulation
    - – binding together data and the functions that manipulate them
  + Inheritance
    - – an entity can inherit attributes from another entity and allows a programmer to create similar entities without needing to redefine similar attributes over and over
  + Polymorphism
    - -- “having many forms”; ie, overloaded functions
  + Abstraction
    - The process of hiding all but the relevant information about a thing to make things less complex and more efficient for the user; allows you to focus on what a thing does instead of how it does it
* Classes
  + Class == blueprint for a ‘house’ design
    - Object == any ‘house’ from the blueprint
      * Instance == a specific ‘house’ (object of the class blueprint)
      * (object and instance often used interchangeably)
* Inline method definition (w default parameter values)
  + Constructor(){datamember = data; datamember2 = data2;}
* Deep copy v shallow copy
  + shallow == bitwise copy; datamember to datamember
  + deep == only need when a classes uses dynamic memory allocation (pointers)
  + memory leak == occurs when ‘new’ operator us used and not followed correctly by a ‘delete’ (typically inside destructor)
    - ‘delete’ operator does not delete the variable but deletes (reallocates/send memory back to heap for use) the address pointed to by the pointer
  + dangling pointer == occurs when pointer is created but not pointed at valid value (address)
    - issue when you try to dereference the pointer (random memory access or null access which will cause errors)
  + Rule of 3 (if we have dynamic memory allocation in a class)
    - 1) create destructor
    - 2) create copy constructor
    - 3) overload assignment operator
* array v pointer notation to access elements
  + array offset notation == array[]
  + pointer notation == \*(ptr + #)
* 4 diff areas of memory
* 
* How to make template
  + template <class ItemType> PlainBox<ItemType>::ClassName(){//default constructor}
* 
* 
* Friends is the only way you can allow one class to access another classes private data members and all of the other data members
* Initializer list
* S() : n(7) {} // constructor definition.
* // ": n(7)" is the initializer list
* // ": n(7) {}" is the function body
* Override versus overload (parameters match identically)
  + Override deals with inheritance
    - Write the same exact function (same parameters and all) in a derived class that the base class already has, so that if you need the derived class to do something different than the base-class’s function call, you are able to do that for the child (derived class)
  + Overload deals with multiple instances of the same function (diff parameters)
    - Same function but each instance has different parameters so that the function can handle many different kinds of calls with various specified information if it has an instance with the parameters passed in
* Virtual methods (use keyword ‘virtual’ in front of the prototype function)
  + Determined at runtime
  + 
  + Pure virtual function
    - Set virtual function equal to 0
    - There is no implementation for pure virtual functions (only a prototype set = to 0)
    - Purpose: need the base class to make the function exist, but the base class will not be the one to create the implementation for it
    - Only way it will be called is if a child class actually writes the function and implementation for it
    - Another purpose = polymorphism
* Unix
  + 
  + d == directory (owner)
  + rwx 🡪 read write and execute permissions
  + rw 🡪 read and write only
  + -rw-rw-r—(user can read and write,
  + – means it is a file if its at start
  + Owner or user, group, other (3 types of users of a file)
  + 
  + 
* 
* Cheapest time to find a problem?
  + Earlier the better (cost of change curve)
* 
  + (phase we implement first needs to be able to stand alone for good soft dev practice)
* 3 error types
  + 1) Compile time (syntax)
  + Runtime error
    - 2) Logic
    - 3) Unexpected error/crash during execution
* CRC
  + 
  + Used to help us design a class
* ADT types
  + LIST
  + QUEUE (PRIORITY QUEUE)
  + STACK
  + HEAP
  + FRIENDS
  + TREES
    - BINARY
    - ADL