Object of Object-Oriented Programming

Terms

1. Object – software bundle consisting of a set of variables which define the states the object can exist in and a set of functions that define the behavior of that object
2. Encapsulation - ability to bind together both data and function in a single unit
3. Data Hiding – hiding the internal state of an object so that its not directly accessible or changeable
4. Data Abstraction – hiding how any of the object's behaviors are performed so that nothing outside the object becomes dependent
5. Inheritance – one class acquires the properties of another
6. Multiple Inheritance – defining a subclass which has more than one parent class
7. Polymorphism – using the same named function in different sub-classes to perform different actions, uses virtual in front of the function name
8. Message – a request sent from one object to another, telling the receiving object to apply one of its methods to itself
9. Instance – the specific realization of any object, such as a car made from a car class
10. Instantiation – process of creating a new object for a class using a keyword
11. Interface File – the .h file that defines the variables (states) and functions (behaviors)
12. Implementation File – the .cpp that defines the details of each of the functions
13. Member access specifiers – public, private, protected
14. Regression Fault – when something new is added that causes the software to not work properly anymore or run slower
15. Virtual Function – allows you to apply polymorphism to a function

Questions

- **An object, is a bundle of what two things?**
  - Set of variables which define the states the objects can exist in
  - Set of functions that define the behavior of that object
- **List and briefly explain the three characteristics of an object**
  - States – member variables in classes
  - Define Behaviors – the methods or functions an object expresses its behavior through
  - Defined ways of modifying the State – none of the fields should be directly modifiable by other objects
- **What is a class and how does it relate to an object in a software system**
  - A class is used to represent objects in programs
- **In the .h file what is the syntax for indicating the class is a sub-class of another class**
  - Class *subclassname* : *inheritance access specifier ParentclassName*
- **In the .h file what is the syntax for indicating the class has more than one parent class (multiple inheritance)**
  - Class *subclassname* : access specifier *Parentclass1Name,* access specifier *Parentclass2Name*
- **Briefly discuss the concept of objects interacting by exchanging messages**

    - Its like the go between for 2 objects. It tells the object what to do rather than thinking that the object needs to call a function. Think like keys, a message is sent to the OS object called Key to say the state is changed to pressed. The key object then sends a message to the OS object called Screen to display the character.

Object Oriented Software Engineering

Terms

1. Structured/Classical development paradigm – includes modular programming, typically produced a design unique to that particular problem
2. Unified Process (Method) – iterative incremental approach to software development, steps are repeated over and over as more detail is added, was adaptable
3. Object Oriented Programming – software is designed around data or objects rather than functions and logic

Questions

- **What are the 3 parts of Modular Programming in the Structured or Classical paradigm approach**
    - Break the problem into several small pieces
    - Solve each problem piece treating it as a new problem
    - Work way down to problems that can be solved directly
- **In Structured Systems Analysis, what was involved in each of the following techniques:**
    - **Logical Data Modeling –** determine data requirements, identify data entities (the objects), and identify the relationships between those entities
    - **Data Flow Modeling –** analysis of how data is input, processed, and moved from module to module within the system, and how data is output
    - **Entity Behavior Modeling –** identify, model, and document events; what events cause changes of state in any of the entities
- **What are the 5 phases of software development in the Waterfall method and what other 2 phases are sometimes added**
    - Requirements Specification Phase, Analysis Phase, Design Phase, Implementation Phase, Testing Phase (Post-Delivery Maintenance and Retirement)
- **What are the 7 steps in the Unified Process**
    - Requirements, Analysis, Design, Implementation (Workflow), Post-delivery maintenance, retirement
- **What are the 4 phases (iterations) in the Unified Process and primary aim of each**
    - Inception – determine if it is worthwhile to develop the target software product
    - Elaboration – refine initial requirements, architecture, monitor risks and refine priorities, redefine the business case, produce software management plan
    - Construction -produce the first operational version of the software
    - Transition – ensure client requirements have been met
- **Match the activities to the correct iteration of the Unified Process they belong to**
    - **Determine if it is worthwhile to develop the target software product -** inception
    - **Ensure the clients requirements have been met -** Transition

    - **Refine the initial requirements, architecture, risks and priorities, redefine the business case -** elaboration
    - **Produce the first operational version of the software –** construction
- **During which Workflow of the Unified Process do you usually write or decide on the following**
    - **Write software development plan, requirements definition and specification -** Requirements
    - **Decide on classes that will be needed as part of the Architectural Design -** Analysis
    - **What member variables, functions will be needed as part of the Detailed Design -** Design

Capability Maturity Model Integrated

- **List and briefly define the 5 levels of capability given for the CMM**
    - Initial – no sound software engineering management practices are in place, usually has time and cost overruns, most responses are in response to crises
    - Repeatable – basic software project management practices, activities based on experience with similar projects
    - Defined – software production is fully documented and defined for managers and technical experts, continually trying to improve the processes
    - Managed – organization sets quantitative, quality, and productivity goals for products and processes,
    - Optimizing – aim is continuous process improvement, knowledge gained through experience is used in future projects
- **Which levels do these activities belong to**
    - **Basic project management processes are established to track cost, schedule, and functionality –** defined
    - **Detailed measures of the software development process and product quality are kept -** repeatable
    - **Continuous process improvement is conducted by using feedback from the software development processes -** optimizing
    - **There are few standards and successes depend on individual efforts and heroics –** initial
    - **The company uses standardized processes for both management and software engineering -** managed

UML

- **What are 3 main parts of UML**
    - Basic building blocks – words of the language
    - Rules controlling how the blocks are put together – the syntax
    - Common mechanisms that apply throughout the language – the conventions or semantics
- **What are the 3 types of Building Blocks**
    - Things – the objects that are parts of a system
    - Relationships – how the objects are interconnected
    - Diagrams – how to combine things to create meaningful pictures

- **What are the 4 types of Things**
    - Structural things – the nouns of the UML models, represent the elements that are conceptual or physical
    - Behavioral – dynamic parts of UML models
    - Grouping – organizations parts, "boxes" into which models can be decomposed
    - Annotational – explanatory parts of UML, used to describe, illuminate, and remark on any element of a model
- **Be able to draw appropriate graphic for Structural, Behavioral, Grouping, Annotational (Things), and Relationship**

Software Requirements and Design

1. Requirements – condition or capability needed by a user to solve a problem or achieve an objective, specific single statement that is testable
2. Requirement Specification – complete description of the behavior of a system to be developed
3. Requirements Analysis – study requirements you have gotten and determine if they are complete and sufficient
4. Step-Wise Refinement
5. Loose-coupling
6. Delegation

Questions

- **4 steps which should be followed in determining the requirements of a system**
    - Elicitation – gather requirements from user, customer, and other stakeholders
    - Analysis – determine what the requirements mean
    - Specification – making a list of requirements
    - Validation – checking with the user to make sure they are correct
- **Good techniques to use when eliciting requirements of a system**
    - Interview with stakeholders – talk to user, customer, and other stakeholders paying attention to what the system must do
    - Questionnaires – create carefully worded questionnaires that ask the client to describe the objects' attributes, behaviors, and interactions in their system
    - User Observation – what how the users are now accomplishing the tasks that the new software should handle
    - Workshops – hold meetings with stakeholders to let them demo and discuss the new system
    - Brainstorming – ask stakeholders to visualize what they would like for the system to do if the sky was the limit
    - Define use cases – basically storyboards of how users will interact with the system
    - Prototyping – create rapid prototypes to get feedback
    - Anticipate future requirements –
- **3 parts required in a Use Case**
    - Clear value – must have clear value to the system
    - Start and Stop – definite starting and ending state
    - External initiator – have something outside the system that initiates the process

- **Explain what the following design principles mean relative to oop**
    - **Identify the aspects of your application that vary and separate them from what stays the same –** for those that don't vary we may want to encapsulate them so that they can't be changed
    - **Program to an interface, not an implementation –** this allows polymorphism to take place, dealing with objects through interfaces allows the ability to get different behaviors without knowing what the actual object is
    - **Favor composition over inheritance –** inheritance has it flaws and poor design in the base classes can cascade down making it harder to correct
    - **Strive for loosely couple designs between objects that interact -** coupling is another level of interdependence, the more an object knows and cares about the implementation of another makes coupling tighter, causes complicated interactions between objects
    - **Classes should be open for extension but closed for modification –** should design in a way that classes can be extended without changing their code

Programming Bits and Pieces

Variables

- **What is an enumerated data type**
    - Provides type-safe data because they can only take on certain values
    - Enum WindDir = {North, Sout, East, West, NoWind}
        - Int WindDir = NoWind
- **What is reference variable**
    - Variable that is an alias for another, once set cannot be changed
        - Int iVar
        - Int& iRef = iVar
- **4 types of type casing**
    - Static_cast – simply changes data type to another, typically numbers or characters
        - Static_cast<dataTypeName>(expression)
        - static_cast<int>(7.9) = 7
    - Reinterpret_cast – crate a value of a new data type with the same bit patter as the expression, used a lot with pointers
        - reinterpret_cast<dataTypeName>(expression)
        - look at slides
    - Const_cast – make a variable value constant or remove the constant quality
        - const_cast<dataTypeName>(expression)
    - Dynamic_cast – safely cast a pointer to a parent class object to a pointer to one of it's subclasses
        - dynamic_cast<dataTypeName>(expression)
            - cShape *s – new cRectangle()
            - cRectangle *r
            - r = dynamic_cas<cRectiangle *>(s)

Functions

- **Function overloading**
  - Two or more functions can have the same name but different parameters. Parameters should be different types, number, sequence of parameters
    - Print(int i), print(double f), print(char const *c)
- **Default arguments**
  - Void somFunction(int x = 0, double d = 1.0, char c = 'a', long l = 1)
  - Allows you to do someFunction() and take all default values or you can input a certain number of arguments and it will override those you give but make the others default
- **Pointers**
  - **Define a pointer to a function given the function arguments and return type**
    - Void RefFunc(int *arg1, double *arg2, char *arg3)
    - RefFunc(&ivar, &dvar, &cvar)
  - **how to set the function pointer pointing to a function of the appropriate type**
    - void foo()
    - func_pointer = &foo
    - func_pointer(arg1, arg2)
  - **use pointer to call a function**
  - **Full summary (alternative)**
    - Some function: void myFunction
    - Declaring: void (*funct_pointer)(int)
    - Initializing: func_pointer = myFunction
    - Invoke: func_pointer(255)

Standard Template Library

- **Do the following for each container used in STL (vector, list, deque, stack, queue)**
  - **Instantiate**
    - vector<int>iVec(5,0) or vector<double>dVec;
    - list<doubl>dList or dList()
    - deque<double>Deque or Deque()
    - stack<datatype> aStack
    - queue<datatype> aQueue
  - **Insert Items/delete/Remove**
    - Vector -Can be done at beginning, middle, or end
      - Insert(itr,newValue) – inserts at specified position
      - Pop_back() -remove elements from back of the vector
      - Erase() – removes elements from a range
      - Push_back() – push elements into a vector from the back
    - List
      - Same as a vector
    - Dequeue
      - Same as vector but can push or remove items from front or back no middle
    - Stack – LIFO

- Push(val)
- Pop

- Queue – FIFO
  - Push(val)
  - Pop()

- **Search/Find**
  - Vector
    - front()
    - back()
    - at(g)
  - List
    - Same as vector
  - Dequeue
    - Would have to iterate through it
  - Stack
    - Can not randomly search, either need to make copy and of the stack and continuously pop to find or do a dequeue
    - Top()
  - Queue
    - Same as stack would have to just pop things off and look
    - Front()
    - Back()
- **Scall all items using for loop**
  - Vector
    - For (auto i = g1.begin(); I != g1.end(); i++
  - List
    - For(list – data.begin(); list != data.end(); i++
      - Cout << list->name
  - Dequeue
    - For(it = g.begin(); it != g.end(); ++it
      - Cout<< *it
  - Stack
    - Would need to empty the stack to accomplish
    - While(!stack.empty())
      - Cout << stack.top()
      - Stack.pop()
  - Queue same as Stack
    - While(!g.empty())
      - Cout << g.front()
      - G.pop()