

From Module To Objects

- It is very difficult to maintain a large monolithic block of code
- The solution is to divide the code into smaller pieces, called modules
- What is a Module?
 - A small and manageable piece of code

From Module To Objects

- “A module is a lexically contiguous sequence of program statement, bounded by boundary elements, having an aggregate identifier”
[Yourdon and Constantine, 1979]
- Examples:
 - Begin...end pairs in Pascal or ADA
 - {.....} pairs in C, C++, or JAVA

From Module To Objects

- Distinction between ACTION, LOGIC, and CONTEXT of a module
 - ACTION
 - What the module does, the behavior
 - LOGIC
 - How the module performs its action
 - CONTEXT
 - The specific usage of that module

From Module To Objects

- In Composite/Structured Design (C/SD) a module should be named after its action not its logic
- Example:
 - A module that calculates the square root should be called **compute square root** and not **m**

Object-Oriented Paradigm

- Major Object-Oriented Concepts
 - Reusability
 - High cohesion and low coupling
 - Encapsulation
 - Data and procedural abstraction
 - Inheritance
 - Polymorphism

Cohesion

- Degree of interaction within a module

- 7 levels of cohesion

- (Good) {
 - Information cohesion
 - » Optimal for Object-Oriented paradigm
 - Functional cohesion
 - » Optimal for Structured programming paradigm
 - (Bad) {
 - Communicational cohesion
 - Procedural cohesion
 - Temporal cohesion
 - Logical Cohesion
 - Coincidental cohesion

Coincidental Cohesion

- A module that performs multiple, completely unrelated actions has coincidental cohesion
 - Reasons for developing coincidental cohesion
 - Management requires 50 executable statements in a module
 - Module is too big or too small
- To correct coincidental cohesion:
 - Break the modules into smaller modules that perform one action

Coincidental Cohesion

- Why is it a bad thing?
 - Degrades the maintainability
 - Corrective or Enhancement maintenance
 - Difficult to understand the product due to its complexity
 - Modules are not reusable. Coincidental cohesion is worse than no modularization [Shneiderman & Mayer, 1975]

Logical Cohesion

- A module that performs a series of related actions, one of which is selected by the calling module
- To correct logical cohesion:
 - Break the modules into smaller modules that perform one action

Logical Cohesion

- Why is it a bad thing?
 - Interface is difficult to understand
 - Degrades the maintainability
 - Corrective or Enhancement maintenance
 - Difficult to understand the product due to its complexity
 - Modules are not reusable

Temporal Cohesion

- A module that performs a series of actions related in time
 - Example: Perform Initialization
 - Open old master file, new master file, transaction file, and print file, initialize sales region table.....
- To correct temporal cohesion:
 - Break the modules into smaller modules that perform one action

Temporal Cohesion

- Why is it a bad thing?
 - Degrades the maintainability
 - Corrective or Enhancement maintenance
 - Difficult to understand the product due to its complexity
 - Modules are not reusable

Procedural Cohesion

- A module that performs a series of actions related by the sequence of steps to be followed
 - Example: Read part number from database and update repair record in maintenance file
- Better than temporal cohesion
 - Actions are related procedurally

Procedural Cohesion

- To correct procedural cohesion:
 - Break the modules into smaller modules that perform one action
- Why is it a bad thing?
 - Degrades the maintainability
 - Corrective or Enhancement maintenance
 - Difficult to understand the product due to its complexity
 - Modules are not reusable

Communicational Cohesion

- A module that performs a series of actions related by the sequence of steps to be followed **and** all the actions are on the same data
 - Example:
 - Update record in database and write it to the audit trail
 - Calculate new trajectory and send it to the printer
- Better than procedural cohesion
 - Actions are more closely connected

Communicational Cohesion

- To correct Communicational cohesion:
 - Break the modules into smaller modules that perform one action
- Why is it a bad thing?
 - Degrades the maintainability
 - Corrective or Enhancement maintenance
 - Difficult to understand the product due to its complexity
 - Modules are not reusable

Functional Cohesion

- A Module that performs exactly one action
 - Examples
 - Calculate sales commission
 - Get temperature of furnace

Functional Cohesion

- Why is it a GOOD thing?
 - Since it only performs one action it is easy to understand
 - Easily maintained
 - Faults can be isolated easily
 - It can be reused
 - A properly designed, documented, and tested module is a valuable asset

Information Cohesion

- A module that performs a number of actions, each with its own entry point, with independent code for each action
- Major difference between logical and informational cohesion
 - Various actions in logical cohesion are intertwined
 - Each actions in information cohesion are completely independent

Coupling

- Degree of interaction between two modules
 - 5 levels of coupling

- (Good)
- Data Coupling
 - Stamp Coupling
 - Control Coupling
 - Common Coupling
- (Bad)
- Content Coupling

Content Coupling

- A module directly references the content of another module
 - Examples
 - Module p modifies a statement in module q
 - Module p refers to local data of module q

Content Coupling

- Why is it a bad thing?
 - Modules are inextricably interlined
 - Inextricable [Webster Dictionary]
 - Forming a maze or tangle from which it is imposable to get free
 - Incapable of being disentangled or untied
 - Difficult to maintain
 - Cannot be reused

Common Coupling

- Modules have access to the same global data
- Why is it a bad thing?
 - Contradicts the structured programming paradigm
 - Modules have side effects that affect their readability
 - Module can be exposed to sensitive data

Common Coupling

- Why is it a bad thing?
 - Difficult to maintain
 - A change in one requires a change in all
 - Difficult to reuse
 - Must reuse all common data

Control Coupling

- A module passes an element of control to another module
- A module directly controls another module
 - Examples
 - Module p calls module q, module q passes not only data but a control that module p has to act on
 - Generally associated with modules with logic cohesion

Control Coupling

- Why is it a bad thing?
 - Modules can not be reused
 - Difficult to understand the structure of modules
 - Difficult to maintain

Stamp Coupling

- Module passes a data structure as an argument to another. The calling module operates only on some component of the data structure
 - Examples
 - Calculate withholdings by passing the employee record
 - The called module does not need the employee's home phone number
 - Security reasons are the main concern

Stamp Coupling

- It is justified to pass a data structure to another module if the called module acts on all the data components of the data structure
 - Examples
 - Invert matrix (old matrix, new metric)
- How to solve this problem?
 - Pass the necessary data to the module

Stamp Coupling

- Performance problems?
 - Don Knuth's laws [Knuth, 1974]
 - First law of optimization: *Don't!*
 - There rarely is a need for optimization of any kind, including for performance reasons
 - Second law of optimization: *Not yet!*
 - Complete the entire product using appropriate software engineering techniques
 - If optimization really is required, make only necessary changes, carefully documenting what is being changed and why.

Data Coupling

- Modules that pass either data structure or simple arguments
- Reduces maintenance
 - Well defined interfaces
 - A change to one module is less likely to effect the other

Important Points

- If a module is tightly coupled with another module, then a change to one module requires a change in the other module
- Well designed code!
 - A good design has high cohesion and low coupling

Data Encapsulation

- A module that contains the data structure together with the actions to be performed on that data structure
- Gathering all aspect of real-world entity together into one unit

Data Encapsulation

- Abstraction
 - The act or process of removing or separating [Webster dictionary]
 - Achieving stepwise refinement by suppressing unnecessary details
 - Separating the detail from design

Abstract Data Type

- A data type, together with the actions performed on instances of that type

Information Hiding

- Data abstraction and procedural abstraction are examples of “Information Hiding” concept
- Data abstraction
 - Hiding the details of the data from other modules
- Procedural abstraction
 - Hiding the details of the procedures from other modules

Information Hiding

- Modules with their implementation details are hidden
- It is more accurate to say “Detail Hiding” versus “Information Hiding”
- This way changing the design of the module will not effect any other module
- JAVA and C++ offer ways to hide these using class specification

Information Hiding

Invisible
to outside

Implementation details of:

queue

queue length

InitializeJobQueue

addJobQueue

removeJobFromQueue

Visible to
outside

Interface information regarding:

initializeJobQueue

addJobToQueue

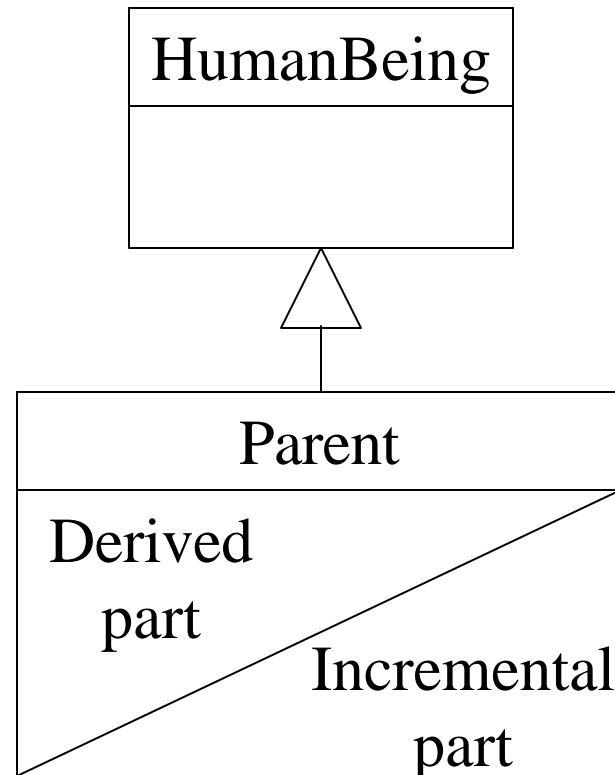
removeJobFromQueue

Objects

- Object is an instance of a class
- Class
 - Abstract Data Type that supports Inheritance
 - Contains data and methods as it's members
 - JAVA
 - A dot (.) denotes membership
 - ClassA.attributeB
 - ClassA.methodC()

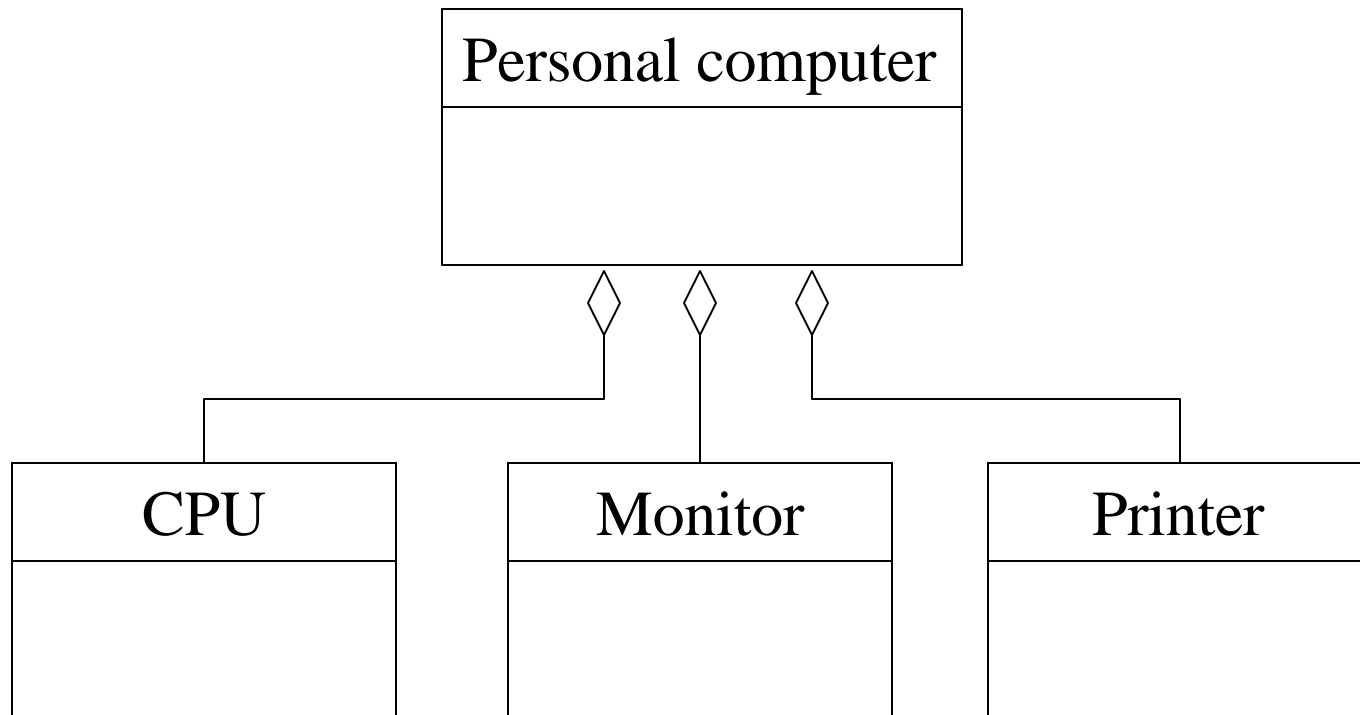
Inheritance

- The idea of extending a data type versus generating a new data type



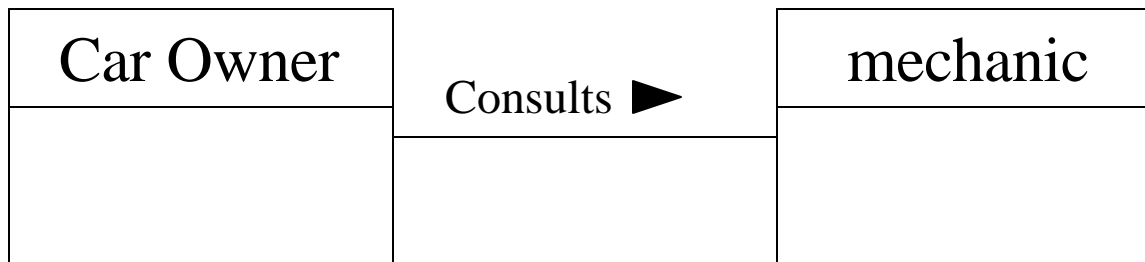
Aggregation

- Components of a class



Association

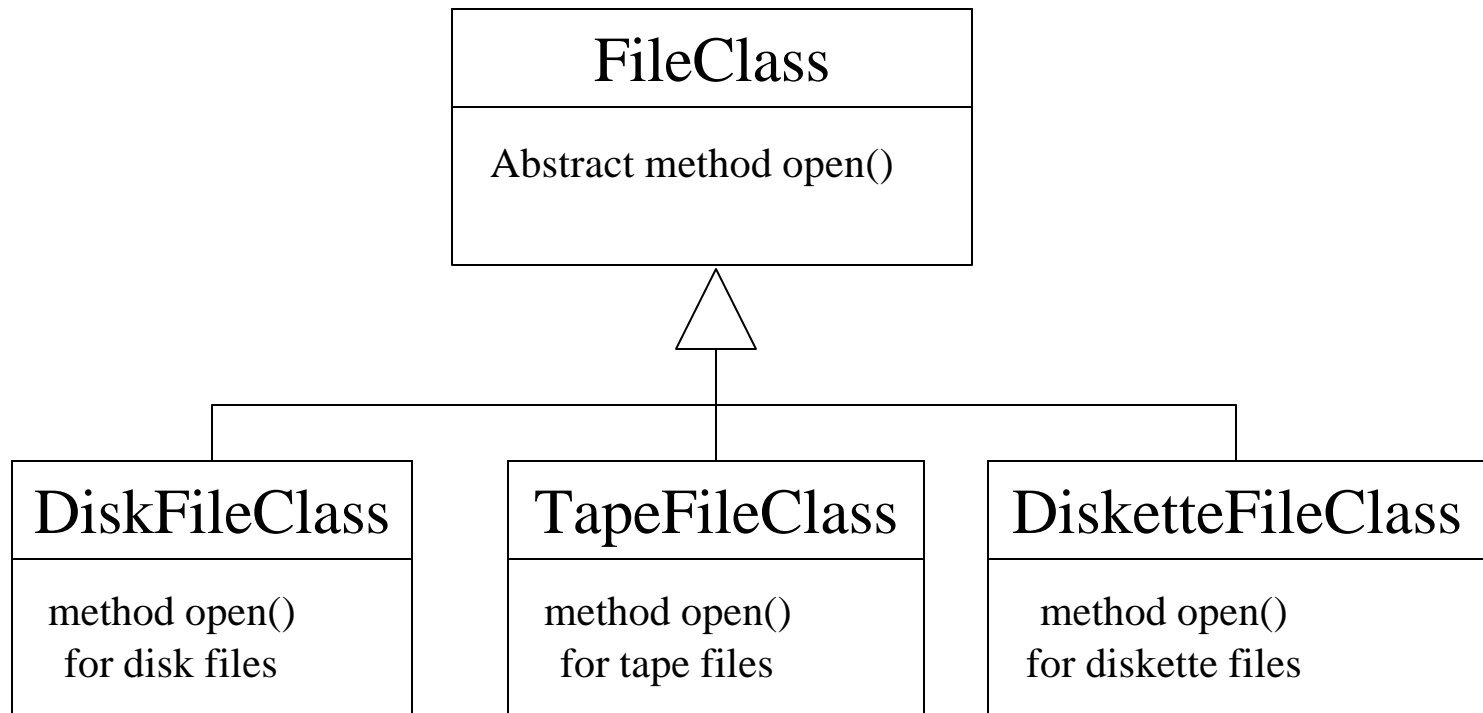
- Some kind of relationship between two apparently unrelated classes



Inheritance, Polymorphism, and Dynamic Binding

- By defining abstract methods (JAVA) or virtual methods (C++) one can delay the selection of an operation
 - Binding is done Dynamically versus Statically
 - Done at run time versus compile time

Inheritance, Polymorphism, and Dynamic Binding



Inheritance, Polymorphism, and Dynamic Binding

- Advantages
 - Can define different methods to handle different types of actions versus writing an “IF” statement in traditional programming languages
 - The programming language/compiler/OS takes care of the decision
 - Binding (act of selecting) is done during RUN time

Inheritance, Polymorphism, and Dynamic Binding

- Disadvantage
 - Impossible to determine at compilation time which version of a polymorphic method will be invoked at run time
 - Could have a negative impact on maintenance
 - Need to understand the product
 - CASE Tools can help reduce the complexity by showing the class tree graphically

Object-Oriented Paradigm

- If Object-Oriented paradigm is a better paradigm, why has the structured paradigm had so many successes?
 - Adopted when software engineering was not widely practiced
 - Software was simply “written”
 - Generating code was the important task for managers

Object-Oriented Paradigm

- Object-Oriented paradigm is an old concept.
- This concept is seen to improve software engineering
- Well designed objects have high cohesion and low coupling
 - Modeling aspects of one physical entity

Object-Oriented Paradigm

- The implementation details are hidden
 - Easily maintainable
 - Communication is performed via messages sent to an object
 - Objects are independent with well defined interfaces
 - Objects are reusable
 - Reusability is enhanced by the concept of inheritance