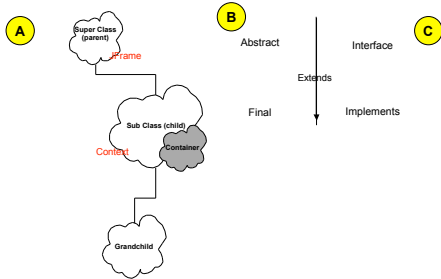


Relationships purpose: architecture



They are just programmer's tools

- If it helps the task to “take on the characteristics” of a Parent Class... then your code “becomes” (is a) the Parent Class, and extends it from there.
- If, however, your code just makes use of another class to do it's job, it just “uses” (has a) another class.

abstract public class Name

- A SuperClass that must be overridden
- Cannot be instantiated
- “Abstract” Dictionary Definition –
 - disassociated from any specific instance
 - difficult to understand
 - insufficiently factual
 - expressing a quality apart from an object
- Used to solve very general problems

Examples of Abstract Programming



- Windowing and menu-building programs (called visual toolkits)
- Database building programs (called engines)
- Frameworks for web sites
- Factory control programs
- Constructing SuperClasses

Final - variables, methods, classes

final public class Name



- A class that can't be extended
- A method that can't be overridden
- A variable that has one, constant value

Interface - *public interface Name*



- An abstract SuperClass, that has only method and variable definitions, and no code.
- If you "implement" an interface, you are obligated to write methods according to the stated definitions.
- Used by Java programmers to set standards:
"If you implement a class based on the following interface, other programmers know the nature of the class behavior"

Interface



```
public interface Constants
{
    public static final int ONE = 1;
    public static final int TWO = 2;
    public static final int THREE = 3;
}

public class Numbers implements Constants
{
    // uses stated values of ONE, TWO, THREE
}
```

Interface - anyone who creates a new shape class must provide these services



```
public interface TetrisShapeClass
{
    public static Appear ( );
    public static RotateCW ( );
    public static RotateCCW ( );
    public static Drop ( int Levels );
    public static Disappear ( );
    public static SlideRight ( int Positions );
    public static SlideLeft( int Positions )
}
}
```

The Object Model



- Classes, Abstract Classes, and Interfaces are used to define intermediate capabilities – methods and variables
- The ultimate Child class finalizes capabilities

Design Time

- Objects instantiated from the Child guarantee a certain type of performance
- Many objects can be created, populating the “solution space” with small computers

Run Time

The Object Model



- “Problem Space” – the real world, where natural conditions occur, and natural problems arise
- “Solution Space” – your program

Problem Space	Solution Space	Abstract Solution Space
Arms, Hands, and Fingers	Automated Machinery	Robots
Operator Choices	Menus	Menu-builders
Letter	Document	Word Processor
Ledger	Accounts Payable and Receivable Program	Spreadsheet
Talking	Email	WWW



Abstraction



- A more generalized solution to a particular problem
- Create a SuperClass that gets a programmer 90%
- Customizable at the SubClass level to solve one problem
- Instantiated into Objects to run

Encapsulation



- “Has a place in it for everything”
- One-stop shopping
- A Class handles all possible problems, issues, boundaries, data, methods, manipulations, around one cohesive task.

Cohesion



- The tendency of a Class to do one, single, independent, highly interrelated job.
- You want strong cohesion.

Bad Cohesion



- A class that does everything
- Grouping behaviors together because they happen “at the same time”.

Functional cohesion



- One function, one job
- Gets only the data it needs
- Produces an output in its simplest form

Example



- An output class - should it handle output to file, screen, and printer, and just take in a flag telling it what to do?
- Or should you write three separate classes, and call the correct one?
- Consider coupling at the same time

Temporal Cohesion



- Behaviors are grouped together because they happen at the same time
- Init routines - *order* is not important, they just all need to be done together.

Sequential Cohesion



- Behaviors are grouped together because they usually happen in order, the output of one behavior being the input to the next

Procedural



- Sequential, but data is not passed between behaviors within the same routine

e.g. Bringing up a system through a step-by-step process (watch your PC boot: disk, ports, network, display, the *order* is important)

Communicational



- Behaviors are grouped because they act on the same data

Logical

- Only a loose association



Coincidental

- No association



Coupling

- A Class' dependency on other classes to do its job.
- You want weak coupling ("de-coupled").



types of coupling



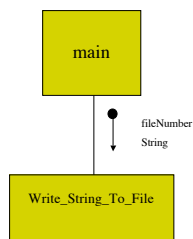
- Two classes/objects use the same static variable
- One class uses methods of another
- One class extends another
- One passes completed data to another
- One pre-processes something, and the other uses it next
- One object passes control information to another

Data Coupling



- A class is self-contained
- Defers to other modules only to get necessary, pre-processed data
- Example?
- `Write_String_To_File(FileNumber, "Hello");`
 - what's coupled? Just called and calling methods, through easily-understood data.

- in "structure chart" form:

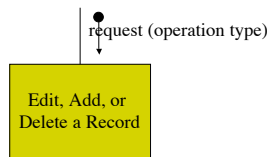


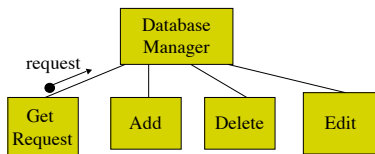
Control Coupling

- A module has all necessary data, but waits until a command comes to tell it what to do
- Flags passed between modules
- Example?
- Look at DbMgr



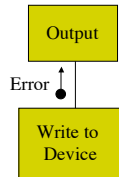
e.g. Db Mgr

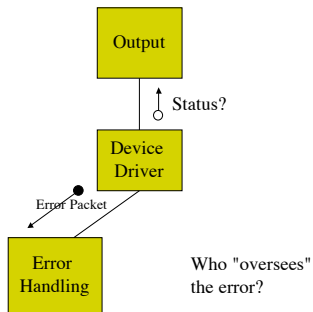




Control Coupling 2

- Module calls another for assistance, and then it receives back an error message
- Inversion of Authority





Common Coupling

- A global (static) variable is shared
- What if - you attempt to fix one module by changing the global variable type?

Stamp Coupling



- A record or array is shared, each class acting on a different part
- Same "what if?"
- The invisible danger of untraceable associations
- Change one, change the record... what happens to all the other modules that depend on it?

Events



- Operator or computer-initiated occurrences that are recognizable by your program
- Operator - Mouse clicks, cursor movement, key presses, drag and drop,
- Computer - Instant Message, clock ticks, communications over the ports
- Programmatic – instantiation, method calls
