

APCS: Plus ça change, plus c'est
la même chose*

Tim Corica

The Peddie School, NJ

NECC, June 24, 2001

tcorica@peddie.org

(* The more things change, the more they remain the same.)

Plan for the talk

- ✓ Find out who you are!
- ✓ The APCS curriculum and exams
- ✓ Languages and APCS
- ✓ History of CS and languages
- ✓ What is OOP?
- ✓ C++, Java, OOP transitions
- ✓ What you can do now

Who are you?

- ✓ Saw my talk last year? _____ Yes _____ No
- ✓ Presently teaching C++: _____ Yes _____ No
- ✓ Presently teaching AP: _____ Yes _____ No
- ✓ Years teaching AP: _____ 0-2 _____ 3 or more
- ✓ Have a CS degree: _____ Yes _____ No
- ✓ Java workshop yesterday? _____ Yes _____ No
- ✓ Studied OOP at some point: _____ Yes _____ No
- ✓ (Who is Tim?)

APCS Curriculum and exams

- ✓ AP Level A: Fundamental programming concepts and skills (CS1?)
- ✓ AP Level AB: Level A, plus dynamic data structures, more depth in algorithm analysis, details of class construction (CS2?)
- ✓ May 2000 about 13,000 A and 7,000 AB exams taken
- ✓ Present language is C++

Languages and APCS

- ✓ Pascal: May 1984-1998 (14 years)
- ✓ C++: May 1999-2003 (4 years)
- ✓ Java: May 2004-???
- ✓ The Java decision: How was it made? Who made it? Ad Hoc committee (Henry Walker, chair)
- ✓ Chris Stephenson's article on Language Choice in JCSE April 2000
- ✓ Don't shoot the messenger!

But, it's not about languages!

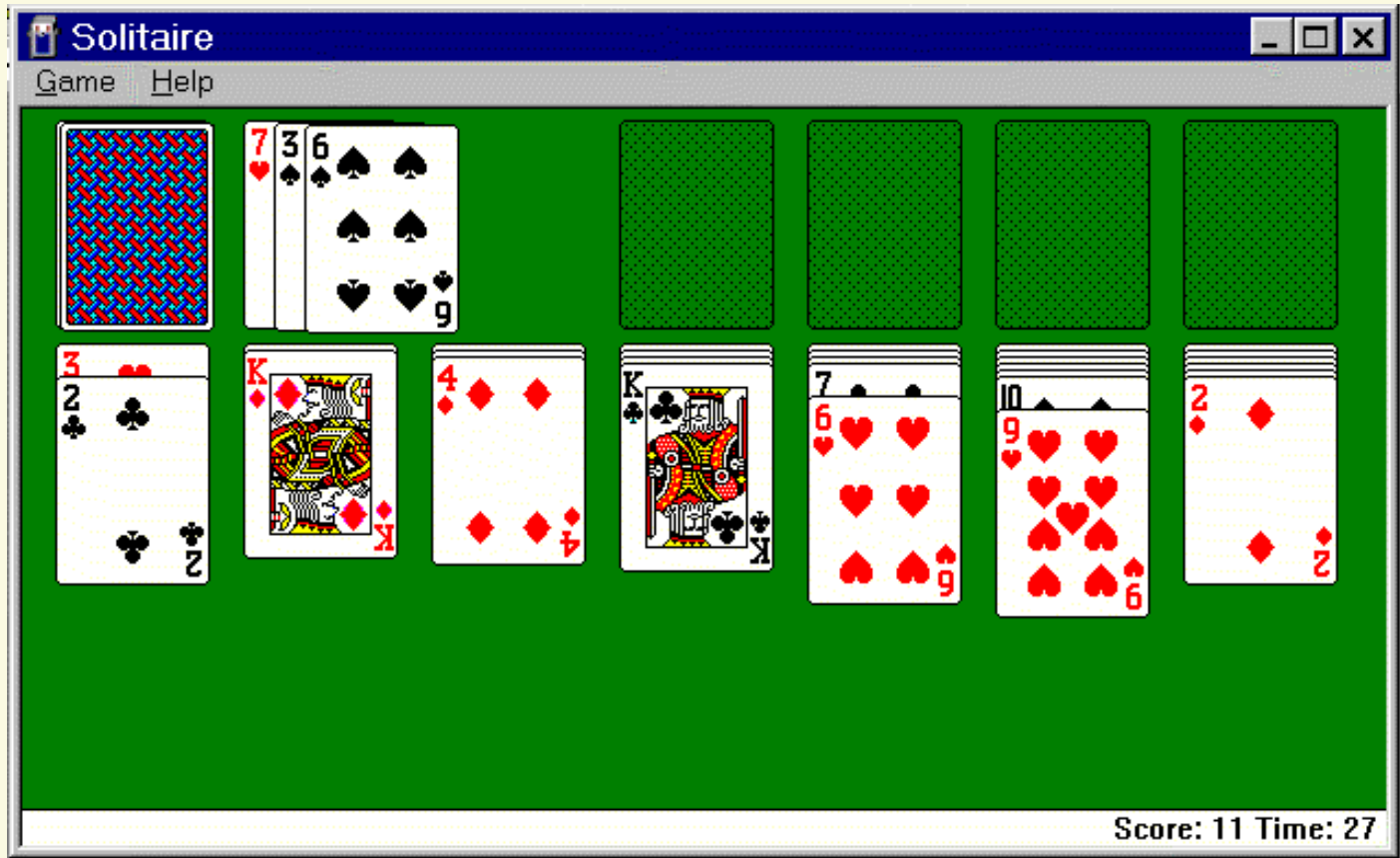
- ✓ 1968: Dijkstra's letter "Go To Statement Considered Harmful", spawns "structured programming"
- ✓ 1977: Jensen/Wirth Pascal: Procedural abstraction (Scheme, Smalltalk start OOP rolling, but remain niche)
- ✓ 1980: Wirth's Modula-2/Ada focus on data abstraction
- ✓ 1985: Stroustrup's C++ makes OOP possible on large scale, but with legacy
- ✓ 1994: Sun's Java is nearly legacy-free, fully_OOP, and meshes with the Web explosion

Brook's "No Silver Bullet"*

- ✓ Programming tools (compilers, IDEs, etc.) are improved nearly to the maximum
- ✓ Remaining software productivity problem is the task of conceptualizing complex systems.
- ✓ After 20 years: OOP may be the closest to a "silver bullet" addressing this issue.

(*From The Mythical Man-Month, 1974 and 1994 editions)

What is OOP?



Alan Kay's five OOP principles

1. Everything is an object.
2. A program is a bunch of objects telling each other what to do by sending messages.
3. Each object has its own memory made up of other objects.
4. Every object has a type (its "class").
5. All objects of a particular type (class) can receive the same messages.

Adapted from Eckel's Thinking in Java 2nd Ed., Prentice Hall, June 2000
(available online at <http://www.eckelobjects.com/>)

Transition to OOP thinking

- ✓ Intelligent use of classes/objects to construct programs
- ✓ Look carefully at the two AP Case Studies:
 - LargeInt: two, hierarchical classes
 - Marine Biology: 6-8 interlocking classes
- ✓ Visual Basic as a starting point
- ✓ Model/View approach
- ✓ Event-driven OOP models (like our graphics chapter).

Transition from C++ to Java

- ✓ Colleges found Pascal→C++ much harder than C++→Java.
- ✓ Statement syntax is nearly identical.
- ✓ Java is inherently more OOP than C++.
- ✓ Java supports strings, range-checked arrays, etc., without adding special libraries!
- ✓ There is a clear and defined “standard”.
- ✓ Once again, a College Board subset

The Java Subset

- ✓ Defines what will be tested, but not what can or should be taught.
 - No I/O of any kind (file or console or GUI)
 - Specific data structures, including ArrayList from Java 2, and many others for AB exam.
 - Inheritance – expanding role of OOP
 - A-level exam: Understand and modify subclasses
 - AB-level exam: Design and implement subclasses

An AP question: AP2001A1

- ✓ Given two classes, where one contains objects of the other, write several member functions of the containing class.
 - Pump class houses data about a gasoline pump.
 - `double GallonsSold();`
 - `void ResetGallonsSold();`
 - Station class represents a collection of pumps.
 - `void ResetAll();`
 - `double TotalSales();`
 - `void CloseStation(ostream & logFile);`
 - private: array of pumps, base price

(a) ResetAll function

✓ C++

```
void Station::ResetAll ()
{
    int k;

    for(k = 0; k < myPumps.length(); k++)
        myPumps[k].ResetGallonsSold();
}
```

✓ Java

```
public void resetAll ()
{
    int k;

    for (k = 0 ; k < myPumps.length ; k++)
        myPumps [k].resetGallonsSold ();
}
```

(b) TotalSales function

✓ C++

```
double Station::TotalSales() const
{
    int k;
    double tot = 0;

    for(k = 0; k < myPumps.length(); k++)
    {
        if(k < 2)
            tot += (myBasePrice + 0.25) * myPumps[k].GallonsSold();
        else
            tot += myBasePrice * myPumps[k].GallonsSold();
    }
    return tot;
}
```

✓ Java

```
public double totalSales ()
{
    int k;
    double tot = 0;

    for (k = 0 ; k < myPumps.length ; k++)
    {
        if (k < 2)
            tot += (myBasePrice + 0.25) * myPumps [k].gallonsSold ();
        else
            tot += myBasePrice * myPumps [k].gallonsSold ();
    }
    return tot;
}
```

(c) CloseStation function

✓ C++

```
void Station::CloseStation(ostream & logFile)
{
    logFile << TotalSales() << endl;
    ResetAll();
}
```

✓ Java

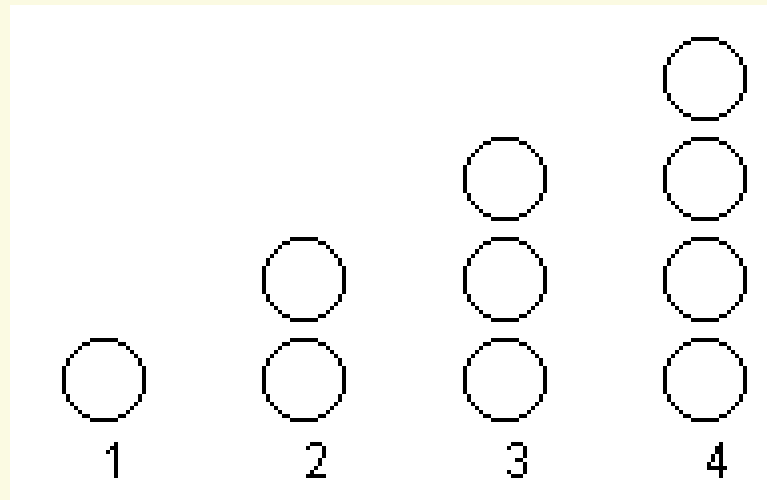
```
public void closeStation (PrintStream logFile)
{
    logFile.println (totalSales ());
    resetAll ();
}
```

Model/View approach

- ✓ Model: Class to house the data
- ✓ View: Class to display the data
- ✓ Controller: Class to interpret user input
 - Sometimes combined with View
- ✓ In early course work, rather than fall back on non-OOP, have teacher write the view and students write the model.
- ✓ Example: Nim

The Nim game

- ✓ Loser is one who takes last stone
- ✓ Must take 1-3 stones from a single column



- ✓ Nehmen is German for "take"

Nim Model

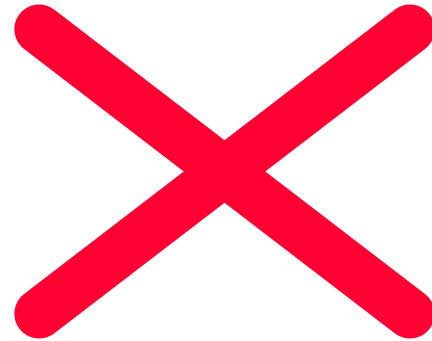
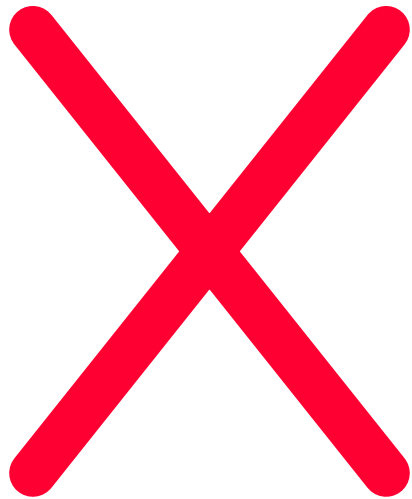
```
class NimModel
{
    /**
     *   Constructor initializes columns to 1-4 stones each
     */
    public NimModel ()
    {}

    /**
     *   Returns number of stones in specified column
     */
    public int numInColumn (int column)
    {}

    /**
     *   Removes stones from column, if a legal move is represented, and returns true.
     *   If the move is not legal, returns false and leaves stones unchanged.
     */
    public boolean take (int column, int numToTake)
    {}

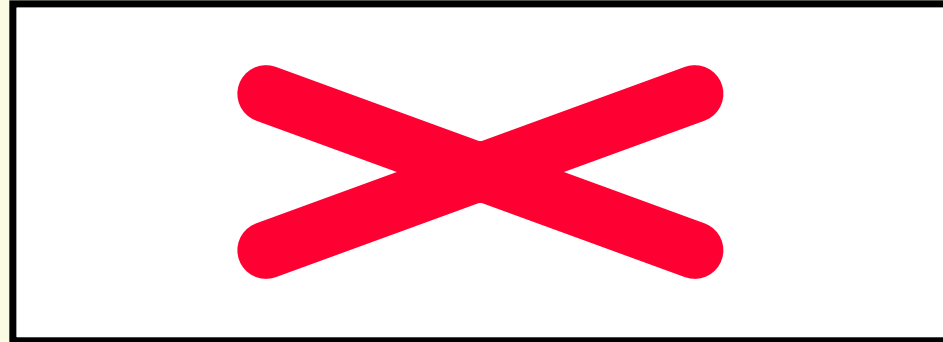
    /**
     *   Returns true if and only if there are no stones remaining
     */
    public boolean gameOver ()
    {}
}
```

Nim model code

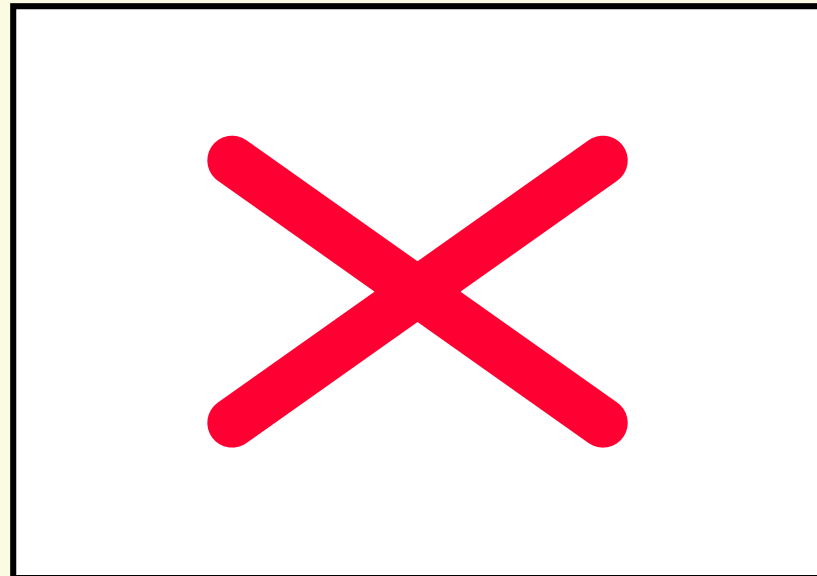


Nim: One Model, two Views

✓ Console



✓ Simple GUI



What can you do now?

- ✓ Add more OOP to your present courses
- ✓ Experiment (or get your kids to experiment) with Java inexpensively
- ✓ Put some Java into your courses - it mixes well with C++
- ✓ Keep an eye out for opportunities for training, etc.
- ✓ *Plus ça change, plus c'est la même chose!*

Some references

✓ Tim Corica: tcorica@peddie.org

✓ This talk:

<http://www.peddie.org/LOCAL/NECC2001.ppt>

✓ College Board APCS page:

<http://www.collegeboard.org/ap/computer-science/>

✓ College Board Java page:

http://www.collegeboard.org/ap/computer-science/html/java_announcement.html

More references

- ✓ Bruce Eckel's Java/C++ page:
<http://www.eckelobjects.com>
- ✓ Mythical Man-Month, by Frederick Brooks
- ✓ Out of Their Minds, by C. Lazere, D. Shasha (Interviews with great CS minds, including Alan Kay)

Java systems

✓ Free/low-cost Java systems

- Ready to Program, <http://www.hsa.com> (low cost)
- JCreator, <http://www.jcreator.com>
- BlueJ, <http://www.bluej.org>
- Sun JDK, <http://java.sun.com> combined with TextPad editor, <http://www.textpad.com>

✓ Higher end commercial systems

- CodeWarrior (Metrowerks, for Mac and PC)
- JBuilder (Borland)
- Visual Studio (MS)
- VisualAge for Java (IBM)