The Year in Review… Changes and Lessons Learned in the Design and Implementation of the AP CS Exam in Java

Scot Drysdale, Dartmouth College
David Reed, Creighton University
Judith Hromcik, Arlington High School
Reg Hahne, Moderator, Atholton High School

Test Development

Scot Drysdale
Dartmouth College
scot@cs.dartmouth.edu

Chair, AP CS Development Committee

So what changed from C++?

- Object Orientation and Polymorphism
  - Inheritance
  - Interfaces
- Collections and Libraries
  - A:
    - ArrayList
  - AB:
    - List (ArrayList, LinkedList)
    - Set (HashSet, TreeSet)
    - Map (HashMap, TreeMap)
    - Iterator, ListIterator
- Design questions (not totally new, but more emphasis)

Collection classes and libraries

- Understand and use Lists, Sets, Maps
- Understand and use iterators
- Understand efficiencies of various implementations
  - ArrayList:
    - set, get are O(1) time worst case.
    - A series of add(k, x) or remove(k) operations are O(length-k) per operation
  - TreeSet, TreeMap
    - set, get, add, remove, contains all O(log n) worst case
  - HashSet, HashMap
    - set, get, add, remove, contains all O(1) expected, O(n) worst case time
  - Iterators
    - O(n) to iterate through entire HashSet or TreeSet
    - TreeSets are iterated in order, O(log n) to get first item

Design

- Given problem description, create a class, instance variables, methods, parameters (with appropriate types)
- Given an inheritance/interface hierarchy, decide at what level instance variables and methods belong.
- Given a problem involving data structures, choose an appropriate structure to meet time/space/functional requirements. (E.g. - search better than O(n), able to iterate through the items in alphabetical order)

2004 Free Response Questions

A1: Word List
- traverse an ArrayList of words, count then remove words of specified length

A2: Pet Parade (Design)
- design and implement classes in a hierarchy (abstract Pet → Cat & Dog → LoudDog)

A3: Pond Stocker (MBS)
- add functionality to class that manipulates the simulation & environment

A4: Robot Cleaner
- traverse an array of items, picking up & moving using a complex algorithm

AB1: Library Items (Design)
- design LibraryItem interface, define LibraryBook class that implements & extends Book

AB2: Approval Voting
- iterate over Sets of votes, create and manipulate a Map of results, analyze efficiency

AB3: Predator Fish (MBS)
- extend Fish class to exhibit new behavior

AB4: Priority Queue
- implement PriorityQueue using a binary search tree, recursive traversal/insertion

What About Java 5.0?

- Will not appear before 2007 Exam
- Current plan is “phase 1” - we will be conservative, but may include more of 5.0 later, if it appears that colleges consider it important.
- Our general philosophy - CS 1 is about teaching Computer Science, not about teaching Java.
- Our main consideration - will the new feature make it easier to ask good questions? Is the benefit worth requiring all AP teachers to include it?
- In free response answers students can use any Java 5.0 feature, so if you want to teach features outside the subset your students can use them.

Generic Collection Classes - Tested

- In Java 1.4, collection classes (List, Set) hold Objects
- Maps relate Object keys to Object values.
- Seldom need (or want!) such generality.

```
// Only Strings stored in conferences
List conferences = new ArrayList();

conferences.add("SIGCSE");
String meeting = (String)conferences.get(0);
```

- Drawbacks: Lots of casting, no compile-time type checking of things added to collections. Can’t tell from the declaration what the collection is supposed to contain.

Generic Collection Class Example

- In Java 5.0, can use a C++ style template to restrict collection classes to hold objects of a given type.

```
List<String> conferences = new ArrayList<String>();

conferences.add("SIGCSE");
String meeting = conferences.get(0);
```

- Eliminates casting
- Get compile-time type checking of things added to collections.
- Self-documenting: can tell from the code that conferences is a list of strings

Why Include Generic Collections?

- Simpler code - less casting
- Clearer code, fewer words needed to describe a collection.

  "stateMap maps state names, each of which is a String, to sets of city names, where each city name is String"

  versus

  "stateMap maps state names to sets of city names"

```
Map<String, Set<String>> stateMap =
    new TreeMap<String, Set<String>>();
```

Writing Generic Classes - Not tested

Writing generic classes involves type parameters, wild cards, and can get very hairy very quickly. We decided that we did not want to test these things. So:

- Use of generic collection classes - yes
- Writing generic classes - no.

Enhanced "for" loop - Tested

A shorthand for using an iterator to run through a collection.

```
String meetings = "";
Iterator iter = conferences.iterator();
while(iter.hasNext()) {
   String con = (String) iter.next();
   meetings += con + " ";
}
```

In Java 5.0:

```
String meetings = "";
for(String con : conferences)
   meetings += con + " ";
```

Why include enhanced "for" loop?

- Simpler, clearer code.
- Lots of questions have loops that iterate through collections (or arrays, which use the same syntax). This will reduce the amount of reading required.
- Avoids "off-by-one" errors when iterating through an array or a List.
- Drawback - does not eliminate the need to teach iterators. Can't do remove, for instance.

Other Java 5.0 Additions - Not tested

- Boxing/Unboxing
- Scanner Class
- Type-Safe Enumerations
- Static Imports
- Variable number of arguments to methods.
- printf
- Other esoteric stuff.

Summary: Java 5.0 tested features

- Generic collection classes (use only)
- Enhanced “for” loop

Test Grading

David Reed
Creighton University
davereed@creighton.edu

Chief Reader for AP CS

The Java Switch

the switch to Java in 2004 went relatively smoothly

- concerted effort was made to prepare teachers & students
- overall student performance was comparable to previous years
- slight drop in number of exams

  2004: 14,337 A + 6,077 AB = 20,414 exams

  2003: 14,674 A + 7,071 AB = 21,745 exams

for the first year of Java, a comparability study was performed

- 14 colleges and universities administered sections of the exam in courses
- they reported each student's grade on the exam, and final course grade
- the correlation of exam score to college grade contributed to AP grade setting

Grading issues with Java

all questions were designed with the APCS Java subset in mind

- however, solutions that utilized constructs/classes outside the subset were NOT penalized (and won't be unless a question specifically forbids it)
- for the most part, students stayed within the subset

as in previous years, some minor errors are ignored when grading

e.g., missing semicolons, = instead of == , case discrepancies

no penalty if fail to downcast when accessing a collection

```
String word = wordList.get(i);
```
instead of
```
String word = (String)wordList.get(i);
```

no penalty if fail to convert between primitive and wrapper class

```
counters.set(i, counters.get(i)+1);
```
instead of
```
counters.set(i, new Integer(((Integer)counters.get(i)).intValue()+1));
```

TEACHERS: KEEP STUDENTS WITHIN THE SUBSET!

OOP emphasis

with Java, object-oriented techniques are now emphasized

- all problems utilized class design and/or implementation
- most problems utilized Java collections, class use
- A2, A3, AB1, AB3 utilized inheritance
- AB1, AB4 utilized interface design and implementation

students did reasonably well, some confusion on OOP concepts

- common error: not understanding class vs. interface
  - e.g., interface definition with instance variables and constructor
  - e.g., `Set s = new Set();` instead of `Set s = new HashSet();`
- common error: not recognizing when inherited data/methods could be used
  - e.g., overriding parent class instance variables & methods, failure to call super

TEACHERS: EMPHASIZE OOP CONCEPTS!

Design Questions

2004 placed a greater emphasis on design

- A2 involved designing & implementing classes in a hierarchy
- AB1 involved designing an interface, implementing it & extending a class

students did well (note: very little algorithmic complexity)

- design questions were 2nd highest averages on both exams
- common errors: described under OOP

	mean score*	% of 0/-	mean w/o 0/-
A1	5.84	18.1%	7.13
A2	5.21	5.5%	5.51
A3	4.04	16.3%	4.82
A4	4.38	23.6%	5.73

	mean score*	% of 0/-	mean w/o 0/-
AB1	5.23	4.8%	5.50
AB2	4.05	16.9%	4.88
AB3	5.60	5.7%	5.94
AB4	3.65	20.8%	4.61

*some tables are based on 2004 preliminary data

TEACHERS: BE AWARE OF DESIGN!

Java Collections

A: array, ArrayList

AB: List, Map, Set, Iterator, ListIterator, Stack, Queue, PriorityQueue interfaces
ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet, ListNode, TreeNode

probably poorest performance on the AB exam

- AB2 involved Maps, Sets, Iterators
- AB4 involved Priority Queues, Trees, recursion
- common error: confused access to collections
  e.g., `wordArrayList[i]` instead of `wordArrayList.get(i)`
- common error: not knowing efficiency of collections
  e.g., for AB2, had to know HashSet access O(1), TreeSet access O(log n)
- common error: unfamiliarity with iterators

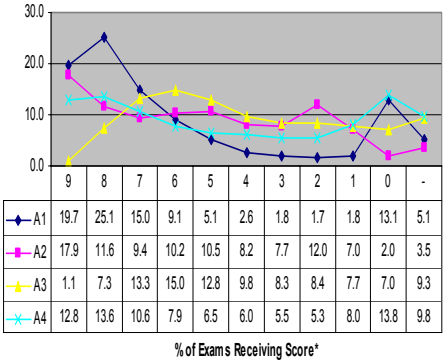
	mean score*	% of 0/-	mean w/o 0/-
AB1	5.23	4.8%	5.50
AB2	4.05	16.9%	4.88
AB3	5.60	5.7%	5.94
AB4	3.65	20.8%	4.61

AB TEACHERS: EMPHASIZE COLLECTIONS!

Java Marine Biology Case Study

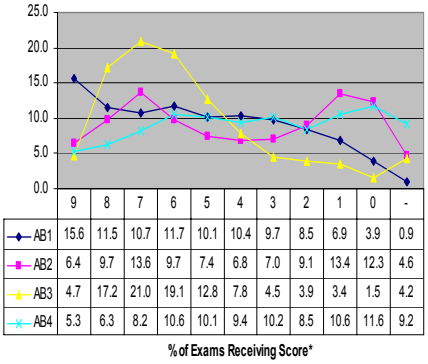
good news: more students (especially AB) knew the case study

- far fewer blanks and zeros than in previous years, especially on AB
- A4 had more blanks than A3; AB2 and AB4 had more blanks than AB3
- AB3 had highest average score on AB exam: 5.6



	9	8	7	6	5	4	3	2	1	0	-
A1	19.7	25.1	15.0	9.1	5.1	2.6	1.8	1.7	1.8	13.1	5.1
A2	17.9	11.6	9.4	10.2	10.5	8.2	7.7	12.0	7.0	2.0	3.5
A3	1.1	7.3	13.3	15.0	12.8	9.8	8.3	8.4	7.7	7.0	9.3
A4	12.8	13.6	10.6	7.9	6.5	6.0	5.5	5.3	8.0	13.8	9.8

% of Exams Receiving Score*



	9	8	7	6	5	4	3	2	1	0	-
AB1	15.6	11.5	10.7	11.7	10.1	10.4	9.7	8.5	6.9	3.9	0.9
AB2	6.4	9.7	13.6	9.7	7.4	6.8	7.0	9.1	13.4	12.3	4.6
AB3	4.7	17.2	21.0	19.1	12.8	7.8	4.5	3.9	3.4	1.5	4.2
AB4	5.3	6.3	8.2	10.6	10.1	9.4	10.2	8.5	10.6	11.6	9.2

% of Exams Receiving Score*

Scoring Distributions

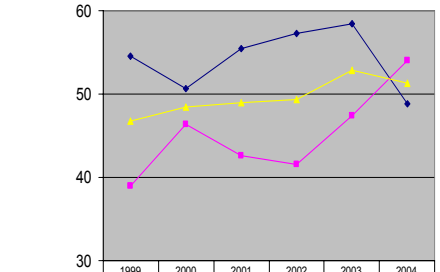
- A exam performance was comparable to recent years
- AB exam performance was slightly worse (bigger change in exam?)

	APCS A Exams				APCS AB Exams			
Grade	**2001**	**2002**	**2003**	**2004**	**2001**	**2002**	**2003**	**2004**
5 (Extremely well qualified)	17.6%	19.3%	17.0%	18.5%	34.0%	34.2%	37.6%	27.0%
4 (Well qualified)	25.1%	25.3%	24.3%	23.5%	12.6%	12.5%	13.8%	18.5%
3 (Qualified)	18.0%	18.3%	19.8%	15.2%	28.1%	26.5%	24.5%	17.8%
2 (Possibly qualified)	9.2%	8.8%	9.3%	9.4%	10.0%	10.6%	10.1%	12.1%
1 (No recommendation)	30.0%	28.4%	29.8%	33.4%	15.3%	16.2%	14.0%	24.6%

Comparison: MC vs. FR

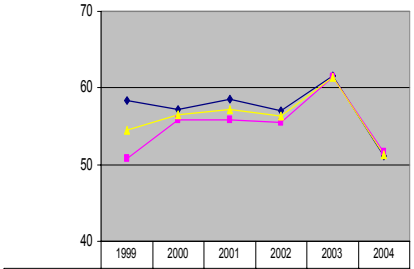
goal: means of multiple choice and free response to be 50% of max

- in 2004, extremely close (51.3% for both exams)
- A exam: free response actually higher percentage (48.8% vs. 54.0%)
- AB exam: multiple choice and free response virtually same (51.1% vs. 51.6%)



	1999	2000	2001	2002	2003	2004
Multiple Choice	54.6	50.6	55.5	57.3	58.5	48.8
Free Response	39	46.4	42.6	41.6	47.4	54
Composite	46.7	48.5	48.9	49.3	52.8	51.3

2004 APCS A Exam: Mean as % of Max Possible Score*



	1999	2000	2001	2002	2003	2004
Multiple Choice	58.3	57.2	58.6	57.1	61.5	51.1
Free Response	50.8	55.8	55.9	55.5	61.4	51.6
Composite	54.5	56.5	57.2	56.3	61.4	51.3

2004 APCS AB Exam: Mean as % of Max Possible Score*

APCS Teaching Tips

Judith Hromcik
Arlington High School
jhromcik@comcast.net

AP CS Development Committee Member

Emphasis on Problem Solving

Problem of the week

- The Puzzling Adventures of Doctor Ecco by Dennis Shasha
  - Lego Problem (paraphrased)
    A rich man wants to build a tower that is 1000 meters high.
    He wants to build the tower in the shortest amount of time possible and has unlimited resources. The tower is to be built with 1 meter high blocks that interlock on top and bottom. It takes 1 day to move a stack of blocks onto another stack of blocks, as long as there are less than 100 blocks in the stack. When stacks contain 100 or more blocks, it takes 2 days. How quickly can this tower be built?
  - Name that Nu mber

Emphasis on Problem Solving

Let students “experience” algorithms before seeing the code

- Searching
  - Scrabble tiles
- Sorting
  - Pancakes
  - Handshakes

Making Abstract Ideas Concrete

Hands-on activities

- Bottle cap arrays
- Objects and references

Stories

- Dr. Seuss
- Other stories and books

Getting Ready for the Exam

- Start introducing AP-style multiple choice questions in your exams as early as possible
- Give mini free response questions on quizzes and tests
- After school reviews
- Saturday Practice Exam

Boxing/Unboxing - Not tested

- The problem: primitives and objects are treated differently in Java, and only objects can be put into a collection.
- Java 1.4 solution: wrapper classes. But lots of "new"s casting, intValue calls, etc. UGLY.
- Java 5.0 solution: still have wrapper classes, but hide them and do the work automatically (mostly).
- Example from last year's exam: voting. Have a Map with the candidate name as the key and the candidate's current vote total as the value. So to count a vote, look up the current count, add one, put it back into the map.

Boxing/Unboxing Example

```
Map tallies = new TreeMap();
…
Integer count = (Integer) tallies.get(candidate);
tallies.put(candidate,
   new Integer(count.intValue() + 1));
```

Java 5.0 solution:

```
Map<String, Integer> tallies =
   new TreeMap<String, Integer>();
…
int count = tallies.get(candidate);
tallies.put(candidate, count+1);
```

If it is so useful, why not include it?

- This was close to a worst-case example - when you want to do arithmetic on something in a collection class. Students will be free to write this solution on future exams.
- Some teachers/professors have valid concerns about teaching boxing/unboxing: it blurs the distinction between primitives and objects, and unintuitive things happen in some cases.
- When writing MC questions we can use Strings or other objects instead of Integers or Doubles, and the problem does not arise.
- Bottom line: we didn't want to force teachers to teach it if we didn't need it and students can use it in free response. No MC question we ask will have a different answer in 1.4 and 5.0.

Scanner Class - Not tested

- A standard I/O class that allows more reasonable input from the console, files.
- We have not tested I/O (other than System.out.print and System.out.println). We haven't missed not being able to read from the console, so why start now?
- We don't want to force teachers who do everything via GUIs to teach the scanner class.

Static Imports - Not tested

- Allows you to use constants like `Color.RED` without typing the "`Color.`" and static methods like `Math.sqrt` without the "`Math.`". So can just say "`RED`" and "`sqrt` " if the following statements are in the program header:

```
import static Color.*;
import static Math.*;
```

- Don't need it, and many consider it to be bad style.

Type-Safe Enumerations: Not tested

- Enumerated types similar to those in Pascal.
- Can be used in switch statements, as keys in maps, and can be printed. Simple and readable.
- Bottom line: they have advantages, but we do not need them to ask the types of questions that we want to ask. Therefore we do not want to force teachers to teach them.