Coming Soon to the AP Computer Science Exam

David Reed
Creighton University, Omaha, NE
davereed@creighton.edu

Ann Shen
Bishop Strachan School, Toronto, Ontario
AShen@bss.on.ca

Laurie White
Mercer University, Macon, GA
white_la@mercer.edu

Agenda

- recent changes in AP Computer Science
- changes for 2007 exam:
  - new Java 5.0 features in subset
  - motivation, examples, what you need to know
- changes for 2008 exam:
  - new case study (GridWorld)
  - motivation, examples, what you need to know
- changes for 2009 and beyond:
  - ???

2

Recent Changes

2003: greater emphasis on student "design"

- design/implement a class given specifications (2003 AB4)
- design/implement classes using inheritance and interfaces (2004 A2 & AB1, 2005 A2, 2006 A2 & AB2)
- design/implement/analyze data structures given specifications (2005 AB2)

2004: switch to Java as programming language

- much stronger emphasis on OO concepts
- use of standard Collection classes: ArrayList, LinkedList, Set, Map,…
- Java Marine Biology Simulation Case Study

3

For 2007: Java 5.0 Features in Subset

Java 5.0 (a.k.a. 1.5) was released in September, 2004

- introduced significant new features, including generics
- many were immediately adopted by CS texts, college courses

APCS Development Committee has taken a conservative approach to adopting features

- is the benefit worth requiring all AP teachers to include it?
- will the feature make it easier to write/answer exam questions?
- note: the APCS Java subset merely defines the *testable* features
  - ✓ a teacher can still teach features not included in the APCS Java subset
  - ✓ a student can use features from outside the subset on the exam

4

Java 5.0 Features in the Subset

- use of generic collections
  - provide compile-time type safety for collections and eliminate the need for *most* typecasts
- enhanced for loop (a.k.a. for-each loop)
  - simpler syntax for iterating over each member of an array or collection
- standardized Stack, Queue, and PriorityQueue
  - generic classes for Stack and PriorityQueue, interface for Queue

5

IN: Generic Collections

in Java 1.4, collections (e.g., `ArrayList`, `Set`) held Objects

```
Set names = new TreeSet();
…
Iterator iter = names.iterator();
while (iter.hasNext()) {
  System.out.println(((String)iter.next()).toUpperCase());
}
```

Java 5.0 introduced generics (similar to C++ templates)

- more transparent data structure w/ compile-time type checking
- less casting required

```
Set<String> names = new TreeSet<String>();
…
Iterator<String> iter = names.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toUpperCase());
}
```

6

Generic Collections (cont.)

generic collection classes will yield better exam questions

- e.g., consider part of a class from the 2005 AB exam

```
public class PostalCodeDB
{                                  // each key is a postal code,
  private Map codeToCityMap;       // its associated value is
                                   // a set of cities
  public PostalCodeDB()
  {
    codeToCityMap = new HashMap();
  }
```

- data structures can be stated less ambiguously using generics

```
public class PostalCodeDB
{
  private Map<String, Set<String>> codeToCityMap;

  public PostalCodeDB()
  {
    codeToCityMap = new HashMap<String, Set<String>>();
  }
```

7

Use but NOT Write

students must be able to use <u>but not write</u> generic classes,

- writing generic classes can get very messy very quickly

```
public class TreeNode<E> {…}
```

vs.

```
public class TreeNode<E extends Comparable<E>> {…}
```

vs.

```
public class TreeNode<E extends Comparable<? super E>> {…}
```

it was decided that having to explain the intricacies of type parameters and wild cards was not worth it (for the exam)

- `ListNode` and `TreeNode` will remain non-generic
- the `Comparable` interface will be used, not `Comparable<E>`

8

Generics vs. Non-generics

students must still be able to read and use non-generic code

- when using `ListNode`, `TreeNode`, or `Comparable`
- when using the MBS case study (for 2007)

students should be comfortable with casting, since examples may appear in Multiple Choice or Free Response code

- however, if they fail to cast when accessing a non-generic collection in a Free Response answer, this is still considered a *non-penalized error*

9

IN: Enhanced For Loop

Java 5.0 introduced the enhanced for loop (a.k.a. for-each)

- cleaner, more abstract notation for accessing each element of an array or collection

from 2006 A2

```
ArrayList productList = new ArrayList();
…
double totalCost = 0.0;
for (int i = 0; i < productList.size(); i++) {
  totalCost += ((Product)productList.get(i)).getPrice();
}
```

vs.

```
ArrayList<Product> productList = new ArrayList<Product>();
…
double totalCost = 0.0;
for (Product p : productList) {
  totalCost += p.getPrice();
}
```

10

Enhanced For Loop (cont.)

the enhanced for loop provides a consistent access pattern for arrays and collections

- it also *reduces* the need for iterators

```
public Set removeSynonym(String syn)
{
  Set affectedWords = new TreeSet();
  Set allWords = wordMap.keySet();
  Iterator iter = allWords.iterator();
  while (iter.hasNext()) {
    String nextWord = (String)iter.next();
    Set synonyms = (Set)wordMap.get(nextWord);
    if (synonyms.remove(syn)) {
      affectedWords.add(nextWord);
    }
  }
  return affectedWords;
}
```

from 2006 AB1

can replace all three lines with a single line

```
for (String nextWord : allWords) {
```

11

Enhanced For Loop (cont.)

note: students still need to know indexing and iterators

- the enhanced for loop won't help if you want to skip elements, access in a different order, or insert/remove/replace elements while traversing

from 2006 A1

```
public void clearConflicts(Appointment appt)
{
  for (int i = apptList.size()-1; i >= 0; i--)
    if (appt.conflictsWith((Appointment)apptList.get(i))) {
      apptList.remove(i);
    }
  }
}
```

- while the enhanced for loop is fine for many simple traversals, students must recognize when indexing or an iterator is needed

12

IN: Stack, Queue, PriorityQueue

starting in 2007, the exam will use standard Java collections

- the use of AP-specific interfaces for Stack, Queue & PriorityQueue was confusing and often conflicted with standard Java classes
- the java.util versions provide a more standard implementation

java.util.Stack<E>: push, pop, peek, isEmpty

```
Stack<String> stk = new Stack<String>();
```

interface java.util.Queue<E>: add, remove, peek, isEmpty

```
Queue<String> q = new LinkedList<String>();
```

class java.util.PriorityQueue<E>: add, remove, peek, isEmpty

```
PriorityQueue<String> pq = new PriorityQueue<String>();
```

13

Stack, Queue, PriorityQueue (cont.)

DANGER: Stack & Queue provide non-standard functionality

- since derived from existing classes/interfaces, they provide methods beyond the classic stack & queue operations

```
Stack<String> names = new Stack<String>();
names.push("Ann");
names.push("Dave");
names.push("Laurie");
…
names.remove("Dave");     // perfectly legal since these
…                         // methods are inherited from
names.add(1, "Gail");     // the Vector class
```

- to test mastery of these abstract data structures, Free Response questions will explicitly limit students to (push, pop, peek, isEmpty) and (add, remove, peek, isEmpty), respectively

14

Minor Subset Changes

the `Random` class has been removed from the subset

- replaced by the general-purpose `Math.random`

expression	range
`Math.random()`	[0.0, 1.0)
`Math.random()*high`	[0.0, high)
`(int)(Math.random()*high)`	0..high-1
`(int)(Math.random()*(high-low+1)+low)`	low..high

for AB, 2 methods have been added to the `List` interface

- both were previously listed in the `ArrayList` class

```
void add(int index, E obj)
E remove(int index)
```

15

OUT: Autoboxing/unboxing

after much debate, autoboxing/unboxing was NOT included

- does allow for primitives to be easily stored in collections

```
Map<String, Integer> words = new HashMap<String, Integer>();
words.put("foo", new Integer(0));
…
words.put("foo", new Integer(words.get("foo").intValue()+1));
```

vs.

```
Map<String, Integer> words = new HashMap<String, Integer>();
words.put("foo", 0);
…
words.put("foo", words.get("foo")+1);
```

while beneficial, there are many subtle conversion rules

- applications requiring autoboxing/unboxing can easily be avoided
- since not needed for the exam, it will not be required

16

OUT: Other Java 5.0 Additions

while potentially useful, other features are not central to APCS

- Scanner, printf (note: subset does not include any input methods)

```
Scanner input = new Scanner(System.in);
while (input.hasNext()) {
  String word = input.next();
  System.out.printf("%10s: %2d\n", word, word.length());
}
```

- type-safe enumerations

```
public enum Response { YES, NO, UNLIKELY, PROBABLY };
```

- methods with variable arguments

```
public static double average(double... values)
{
  double sum = 0;
  for (double v : values) sum += v;
  return sum / values.length;
}
```

- static imports, annotations, …

17

Again: OUT != BAD

just because a feature is not included in the APCS subset doesn't mean it's bad or that teachers shouldn't cover it

- it simply means those features will not be tested on the exam
- it is expected that teachers will cover some of these features

  e.g., `Scanner`, `printf`, simple autoboxing/unboxing
- students are free to use any of these feature in writing free response solutions

  caveat: readers are human, so esoteric code runs a risk

some features (e.g., autoboxing/unboxing) may be revisited as college practices become more uniform

18

FYI

see Cay Horstmann's article on AP Central for more detailed information and examples

- *Teaching with Tiger: Using Java 5.0 Features in AP Computer Science Courses*, by Cay Horstmann
- http://apcentral.collegeboard.com/members/article/1,3046,151-165-0-49154,00.html

19

For 2008: GridWorld Case Study

Fourth major reworking of the case study.

GridWorld goes back to basics, serving as a code framework for test questions.

As a result, GridWorld is significantly smaller than MBS.

GridWorld is designed to be more flexible and easier to use earlier in the course.

Less material is provided by the committee, more is expected from the community.

20

Lessons Learned from MBS

- visual framework is very motivating
- classes were flexible; supported many exam questions
- teachers took advantage of extension points
- complexity was a drawback

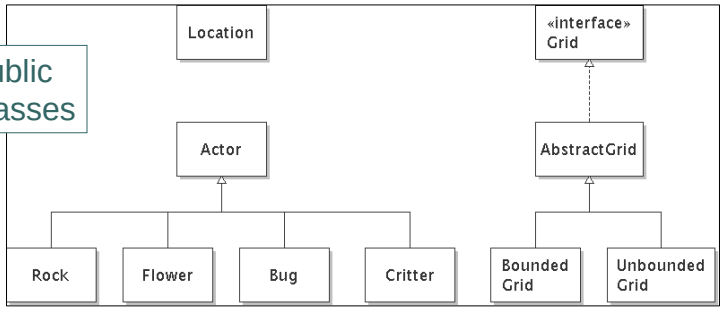
themes for new case study

- continuity
- simplicity
- testability
- extensibility

Overview

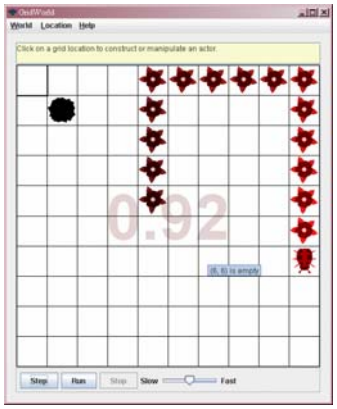
- framework derived from MBS (GNU license!)
- fewer classes/interfaces: 4 (A) / 7 (AB) implementations, 5 API's
- simplified API: easier to remember, easier to formulate questions
- layers: use framework at multiple points in your course
- much easier to add your own classes

public classes



Layer 1: Objects

turtle-like graphics: bug drops flowers



direct manipulation (like in BlueJ)

Can be used as early motivation for objects

23

Layer 1: Objects

- a first look at GridWorld
- exploring Actor state and behavior
- demo:
  - Bug (testable code)

24

Layer 2: Inheritance

- simple inheritance
- three basic methods:
  - canMove()
  - move()
  - turn()
- demo:
  - BoxBug (testable code)

25

Layer 3: Interacting Objects

- uses template method, like `Fish.act`
- override methods to define how critters find neighbors
  - find neighbors
  - process neighbors
  - find candidates for move locations
  - select a move location
  - make a move
- clean and simple behavior, less randomness than MBS

26

Layer 3: Interacting Objects

- demo:
  - ChameleonCritter (testable code)
  - CrabCritter (extra demo code)

27

Layer 4: Data Structures

- little change from MBS
- uses generics
- `Grid<E>` similar to `Map<Location, E>`
- can contain any type, not just `Locatable`
- `AbstractGrid` example of abstract class
- AB Material

28

Testability

- `Bug` subclasses lend themselves to multiple choice questions:
  - e.g., "Which pattern does `MysteryBug` produce?"
- `Critter` subclasses are easily understood
  - e.g., no confusing breed/die issues
- `Grid` useful for a wide variety of questions

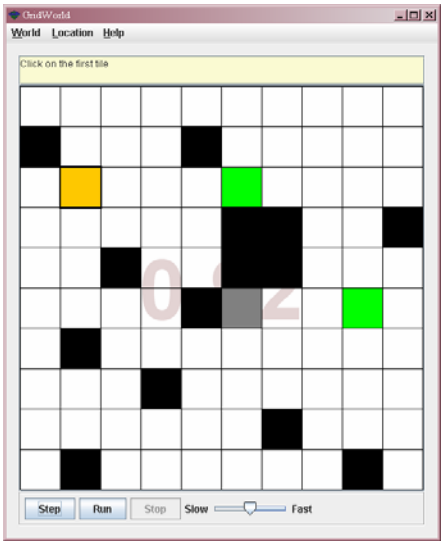
29

Extensibility

- to add new actor, simply supply subclass and GIF image
- image is colored and rotated automatically
- easy to do game worlds (e.g. Memory, Sudoku)
- extensions are optional

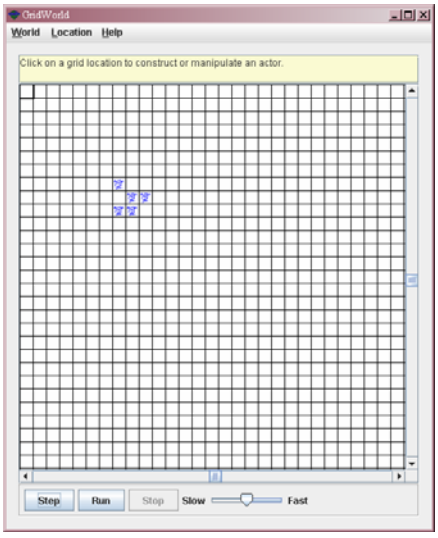
30

An Extension - TileGame



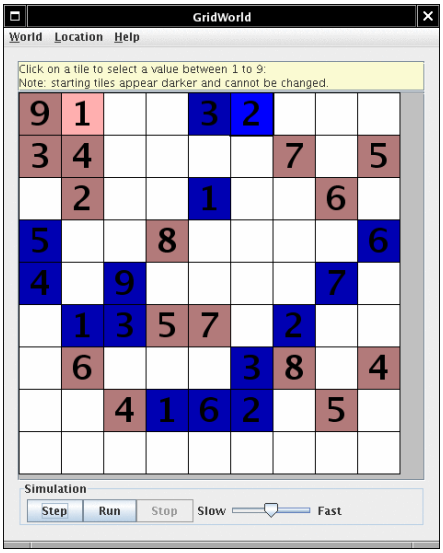
31

An Extension - GameOfLife



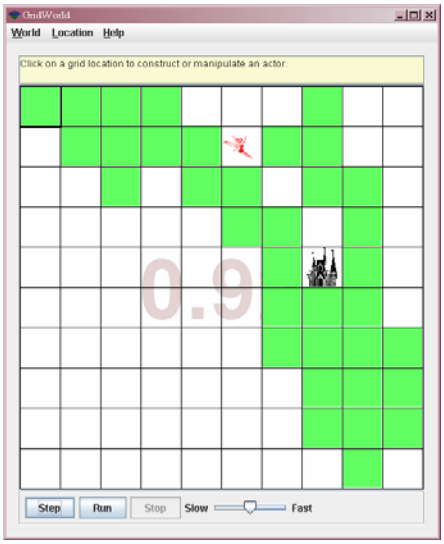
32

An Extension - Sudoku



33

An Extension - APACWorld



34

Schedule

- code and narrative finalized at APCS Development Committee winter meeting
- "Fresh Eyes" review Spring 2006
  - sign up at `http://gridworld.info`
- open to the public Fall 2006
- first use in 2008 exam
- as always, schedule may change...
  - watch AP mailing list, AP Central for announcements

35

For 2009: ???

a college survey is planned for 2006-2007

- will assess current practices and their demands of AP students
- potentially, could affect the required topics in the APCS curriculum
- it is hoped that the size of the A and AB curricula could be reduced
  - many OO/Java features have been added, few concepts removed
  - currently, the APCS curricula contain more material than most colleges

call for discussion:

- what Java/programming features could be moved from A to AB?
- what Java/programming features could be removed from AB?

36