Exam 2
Study Guide

Exam 2 is in class on July 31, 2007

Ok, this study guide is done changing! Note that the code for Asteroids is now on the snarf site -- “05_asteroids” under classwork.

(updates since the guide was originally posted are in green)

If something isn’t at least briefly mentioned in this study guide, then there won’t be any questions on it on the exam. However, it is possible that a few things are mentioned in this guide, but do not appear on the test. Note: Things can be on the test as extra credit even if they are not in this guide.

Frequently Asked Questions

1. • Q: Are these questions really frequently-asked? A: No.

2. • Q: Is the exam comprehensive? A: Yes, sortof. You’ll still have to know what an integer is, for example, etc.

3. • Q: Will the Useless Facts of the Day be on the exam? A: No, but there may be an extra credit question in the same style as last time...

4. • Q: What chapters of the book does the exam cover? A: Everything we’ve talked about in class is in chapters 1 - 10 (but the things not mentioned in this guide will not be on the exam).

5. • Q: Will we have to write more complicated code on this exam than we did last time? A: Yes.

6. • Q: What are all the answers? A: Ahahahahaha.

7. • Q: That wasn’t a very funny joke. A: That’s not a question.

Exam Format

This exam will involve much more game code than the last exam. In particular, many (but not all, and maybe not even the majority) of the questions will be based around the game of Asteroids, which you can snarf in Eclipse (under “classwork,” it is “05_asteroids”). There will be questions asking you to:

1. • find bugs in certain sections of code
2. • simulate what happens when code runs
3. • write code to fix bugs
4. • describe how (and/or write code) to enhance a game / add a feature
5. • explain what certain code does, or write code matching an explanation

Terms to Know

You’ll need to know many of the terms from the last test to understand the questions on this exam, but there won’t be explicit definition-matching like on the last exam. There may, however, be some labeling (of the parts of a for loop, for example...).

Also, know the % (modulus, or “remainder” operator), and know the terms in bold throughout this guide.

Other Things to Know

1. • Debugging
   2. ❖ The difference between a runtime error (a “bug”) and a compile-time (or compilation) error
   3. ❖ A syntax error is a failure to use proper Java syntax (the failure to follow the rules of Java programming -- for example, forgetting a semicolon, or parentheses, or sending the wrong type of argument to a method). Syntax errors are compilation errors. These errors are usually signified by red squiggly underlines in Eclipse.
   4. ❖ A semantic error is when there is a difference between what your code means and what you think it means (in other words, a difference between what your code does and what you think it should do). For example, perhaps you wrote a “for” loop that stopped looping one iteration too soon. Semantic errors are runtime errors.
   5. ❖ Given some (fairly simple) code, be able to “hand-simulate” it and describe what it does. For example, pretend you put a breakpoint in a loop, and write what the values for certain variables would be at each loop step.
   6. ❖ Given code with a bug (a semantic error) and a description of what the faulty behavior of the program is, find the bug and explain how to fix it
7. • Collections
   8. ❖ Know the differences between the built-in Java array, and Java’s ArrayList class of objects
   9. - The biggest difference is that you can add things and delete things from an ArrayList, changing its size dynamically. You have to re-create an entire array, on the other hand,
if you want to change its size, since you can only set the size when you create it.

- You might use an **ArrayList** if you need more flexibility in how many things are in the list while the program is running -- for example, if you don’t know how many things will be in the list when you create it, and instead you just want to create them and add them as the program goes on.

- Example of where an ArrayList makes sense: the FANG engine stores GameLevels in an ArrayList, and it stores the Sprites on the canvas in something similar, called a LinkedHashSet (like an ArrayList, this also grows and shrinks dynamically as you add and remove things) -- this is because the game engine has no idea how many sprites you are going to eventually want to add to the canvas; you just tell it each time you add one!

- Be able to write code to create a collection, add items to it, and access items in it (indexing)

- Know that you should normally use things like `blocks.length`, or `blocks.size()` instead of a number, and know why

---

### Loops

- Label the four parts of a loop: `initialization`, `test (condition)`, `update`, and `body`

- Also, know what each part does

- Know that **while** and **for** loops are really equivalent in power; it’s up to the programmer to choose which one to use based on convenience

- Be able to write actual code for loops that...

  - position a collection of Sprites in a simple pattern (like the grid in Breakout)

  - determine an extreme value in a collection (for example, a min or max value)

  - calculate a value (for example, sum the first twelve multiples of three)

---

### Colors

- Know that Colors in Java are made by combining the primary colors red, green, and blue

- How do you make white? or black? or grey? or purple?

- Know the range of values for each color

- How would you make a random color?

- How could you use a loop to create a range of colors and store them in a collection (a series of progressively lighter shades of red, for example)?

- Know how to use a color palette (a collection of colors) to assign different colors to each of a bunch of Sprites in a collection (where perhaps the collection of Sprites is larger than...
29. ❖ You don’t have to memorize how to make a Sound, an ImageSprite, or a help file.

30. • Searching and Sorting

31. ❖ Hashing will not be on the test (except possibly as extra credit)

32. ❖ If we doubled the size of a list you were searching for something in, how would that change the time it would take to find the thing using the linear search algorithm (in the worst case)? How about the binary search algorithm?

33. ❖ If an algorithm takes 5 seconds to sort 10 things, and it takes 10 seconds to sort 20 things, what is its performance? -- O(N), or proportional to the number of things in the list -- because if we double the number of things, then the time it takes also doubles.

34. - Similarly, O(N^2), or proportional to the square of the number of things in the list, means that if we double the number of things, then the time the algorithm takes quadruples (double squared = quadruple)

35. - O(log (N)), or proportional to the base-2 log of the number of things in the list, means that if we double the # of things, then the time the algorithm takes only increases by one (for example, log 8 = log 2^3 = 3, log 16 = 4, log 32 = 5, log 64 = 6, etc.).

36. - So, from fastest to slowest (O(log N) is the fastest):
   O(log (N)) < O(N) < O(N^2)

37. ❖ Write out the binary search algorithm in English

38. ❖ Be able to do a binary search on a short list of numbers

39. ❖ Can you do binary search on unsorted lists? What kind(s) of search can you do on unsorted lists?

40. ❖ Write out in English what “O(N^2),” or “O(N),” or “O(log(N))” means. For example, O(N^2) means “proportional to the square of the number of things in the list”

41. ❖ Name one slow and one fast sorting algorithm

42. • Levels (we’ll discuss this stuff on Monday)

43. ❖ Given a game’s code, what aspects of the game play change with each new level (ex., are there more asteroids? do they move faster? etc.)?

44. ❖ Write a line of code to change an aspect of a game when the level changes.