The final exam for this course is scheduled for Thursday May 6, at 7:00 PM. It will be held in our regular classroom. The exam will be five pages long and will take most people less than one-and-a-half hours, with a time limit of two hours.

The exam is cumulative, with some emphasis on material that was covered since the second test. The final exam counts for 15% of your grade for the course.

The first four pages of the exam will be similar to previous tests, with a mix of definitions and short-essay questions, programming problems, and questions that ask you to understand and interpret Java code. You will not be asked to write any sorting subroutine, or to work with GUI code except possibly for basic drawing on a Canvas.

The last question on the exam will be a full-page, twenty-point essay question on the topic of program development and how it is supported by the Java programming language. Topics related to program development include pseudocode, step-wise refinement, subroutines, top-down and bottomup design, object-oriented programming, design of classes and class hierarchies, state variables, and the design of event-driven GUI programs. You should be prepared to write a coherent and thoughtful essay on this topic.

Don't forget that solutions to tests, quizzes, and most of the labs can be found on the course web page. And answers to the end-of-chapter quizzes and exercises from the textbook are also available online.

Some things that have been covered since the second test:

extending a class; subclasses and superclasses inheritance class hierarchies polymorphism object-oriented programming class Object; every class is a direct or indirect subclass of Object abstract classes and abstract methods interfaces (i.e. the Java reserved word "interface") the special variable "this" using "this" to access member variables hidden by local variables the special variable "super" using "super" to call a method from the superclass nested classes GUI programming: components, layout, and events for-each loops; using for-each loops with arrays and ArrayLists the search problem for arrays the linear search algorithm sorted arrays; what it means for an array to be sorted the basic idea behind the binary search algorithm for sorted arrays how to think about the efficiency of an algorithm why binary search is so much faster than linear search the basic idea of the the Selection Sort algorithm the basic idea of the Merge Sort algorithm why Merge Sort is so much faster than Selection Sort the structure of two-dimensional arrays; how they are stored in memory processing two-dimensional arrays

Important topics from earlier in the course:

algorithm

machine language, high-level programming languages, and compilers syntax and semantics of programming languages public static void main(String[] args) literals, variables, expressions, and operators Java's primitive types, including int, double, boolean, and char System.out.print(x), System.out.println(), and System.out.println(x) TextIO methods: TextIO.getln(), TextIO.getlnInt(), TextIO.getlnDouble() Math.random(); using Math.random() to make random integers the String type; Strings are objects String methods: str.length(), str.charAt(i), str.equals(s1), str.indexOf(ch) control statements: if, while, for exceptions and the try..catch statement how to throw an exception, and why you might want to do so programming style rules and why they are important arrays base type of an array elements of an array; referring to array elements as for example A[i] using "new" to create arrays, for example: new int[10]basic array processing such as summing, counting, finding a max two-dimensional arrays; using nested for loops with two-dimensional arrays drawing on a Canvas using a GraphicsContext, g basic color constants like Color.RED, Color.BLUE, Color.BLACK setting drawing colors with g.setFill(c) and g.setStroke(c) drawing subroutines: g.strokeLine, g.strokeRect, g.fillRect, g.strokeOval, g.fillOval subroutines and parameters; formal parameters vs. actual parameters black boxes; separation of interface from implementation the access modifiers "public" and "private" return types and the return statement; void the dual nature of classes: the static part and the non-static part scope of a variable; local variables vs. global variables named constants and the "final" modifier; why named constants are used packages; importing classes from packages top-down and bottom-up design classes and objects instance methods and instance variables instance variables represent "state" of objects; methods represent "behavior" getter and setter methods and why they are used pointers, also known as references; the special value null assignment and equality-testing for objects constructors; calling a constructor with "new"; writing a constructor for a class

Some sample questions from old final exams.

Here are some questions from final exams that I have given in the past in CPSC 124. This is not meant to cover every possible topic that might be on the exam this year, but it will give you some idea of what types of questions might be. Note that the average difficulty of the questions on the exam will be lower than the sample of questions given here.

1. Some short programming exercises...

a) Use a for loop to print out all the multiples of 5 from 5 to 100, with each number on a separate line.

b) Write a program segment that gets an integer from the user in the range 1 to 100 (inclusive). Use a **while** loop to make sure that the number that you get is actually in the specified range. (You can use *TextIO* for input.)

c) Write a program segment that simulates rolling a pair of dice 1,000,000 times and counts how many times the dice come up doubles (that is, how many times the values on the two dice are the same).

d) Write a code segment to simulate the following experiment: Toss a coin over and over, until it has come up *heads* ten times. At the end, print the number of times that the coin has come up *tails*.

2. Show the output of each of the following program segments:

a)	<pre>int x,y;</pre>	b)	<pre>int[] A,B;</pre>
	x = 100;		A = new int[5];
	y = 0;		B = new int[5];
	while $(x > 0)$ {		A[0] = 1;
	x = x / 2;		B[0] = 0;
	<pre>System.out.println(x);</pre>		for (int i = 1; i < 5; i++) {
	y++;		A[i] = 2 * A[i-1] + 1;
	}		B[i] = A[i] + B[i-1];
	<pre>System.out.println(y);</pre>		<pre>System.out.prilnln(A[i] + " " + B[i]);</pre>
			}

3. The following code, although syntactically correct, has two semantic errors that would cause exceptions at run time. The intent is to create five buttons that are initially disabled. Find the two errors and state the problem in each case.

```
Button[] buttons;
buttons = new Button[5];
for (int i = 0; i <= 5; i++) {
    buttons[i].setDisable(false);
}
```

4. Writing subroutines...

a) Write a subroutine to count the number of times that the number 17 occurs in an array of type *int*[]. (The array should be a parameter; the return value is the count.)

b) Write a subroutine named *containsAll* with return type *boolean* and two parameters of type *String*. The value of *containsAll(str, chars)* should be *true* if the

string *str* contains **every** character in the string *chars*. (You will need to use either nested *for* loops or the string method *indexOf*.)

c) Write a subroutine that will strip extra spaces from a string. The parameter of the subroutine is a *String*. The return value is the same string, except that every substring of consecutive spaces has been replaced by a single space. For example, using __ to represent a space, if the parameter is "Goodby____reruel__world", then the value returned by the subroutine is "Goodby__cruel_world". (Hint: Only include a space in the output string if the preceding character is not also a space.)

5. Suppose that *Document* is a class that represents a word-processing document. This class has an instance method

```
public void print()
```

that prints the document. Write a **complete subclass** of class *Document* that has exactly the same behavior, except that it also keeps track of the number of times that the document has been printed (that is, the number of times that *print()* has been called). The class that you write should have a method *timesPrinted()* that returns the number of times that the document has been printed. (Hint: You will need to use **super**.)

6. Consider the following arrays:

```
String[] cityNames = new String[50]; // Names of 50 cities.
String[] birdNames = new String[100]; // Names of 100 bird species.
int[][] population = new int[50][100]; // Population of each bird in each city.
```

Assume that the arrays have already been filled with data. The array *population* contains data about the population of each of the 100 species of birds in each of the 50 cities. That is, population[b][c] is the number of birds of species number b that live in city number c.

a) Write a code segment that will add up all the numbers in the *population* array (giving the total number of birds of all species in all cities).

b) Write a code segment that will do the following for each city: **Count** the number of bird species that are found in that city (that is, the number of species for which the population is greater than zero). Then **print** the name of the city along with the number of bird species found in that city.

c) Which species of bird has the largest total population? Write a code segment that determines the answer and prints the **name** of the species.

7. Write a code segment to create the situation shown in the picture, including an array of type *int*[] and a variable A that points to the array. (Include the variable declaration.)



8. Consider the following declaration of the class *Student* and the array *stu*, and assume that the array has **already** been filled with data for 100 students. Write a code segment that prints the names of all students who have an A average, that is the average of their *test1*, *test2*, and *test3* is 90 or above.

```
public class Student {
    public String name;
    public double test1;
    public double test2;
    public double test3;
}
Student[] stu = new Student[100];
```

9. We have used the expression (int)(1+6*Math.random()) to simulate the rolling of a standard 6-sided die, but some games use dice with different numbers of sides.

a) Write a complete Java class that represents a *single die* with any given number of sides. The class should have a constructor with no parameters that creates a standard 6-sided die. It should also have a constructor with one parameter of type *int* that specifies the number of sides of the die; the value of the parameter must be greater than 1. It should have a method for rolling the die and a method for reading the number that is currently showing on the die.

b) A certain game uses five dice, which have 4, 6, 8, 12, and 20 sides respectively. Write Java code that creates five objects belonging to the class from part **a**) to represent these five dice. You can use either five separate variables or an array to hold the data.

c) Write Java code that will roll the five dice that you created in part b) and print the sum of the numbers showing on the five dice.

10. Some short essay questions...

a) Define the terms *syntax* and *semantics* as they relate to programming. Include some examples in your discussion.

b) Some members of classes are *static*, and some are not. Carefully explain the difference.

c) Discuss the *new* operator, why it is necessary, and how it relates to *constructors*.

d) Describe the *binary search* algorithm, and explain briefly why it is so much more efficient than *linear search*.

e) What does it mean for a variable to be *private*, and why should private variables be used?

f) What is an algorithm? What is the difference between an algorithm and a program?

g) What are *pointers*, and where are they used in Java?

11. Write a JavaFX code segment that will draw the picture shown below. Use a *for* loop. There are exactly 11 lines in the picture. The scale that you use is up to you, but your picture should have the same form as the one that is shown. You do not have to set color or line width; just draw the lines.

