CSC 254 Study Guide 1, Fall 2016

Exam Format and Rules

* The exam will be on Wednesday, October 12, 2016
* The exam will be in Remington 105
* The exam will be in Moodle
* There will be mostly objective questions, but there will be some essay, code writing, and short answer.  Essay questions must be answered with full sentences and proper paragraphs.  Do not use bullet points to answer essay questions.
* The exam is closed book  You may not use notes, Eclipse, Netbeans, or other application.
* You must use one of the lab computers and your own Missouri Western login ID
* Non-native speakers of English will receive the following accommodations:
  + Non-native speakers of English may use an English/native language dictionary
  + Paper dictionaries or dedicated translation devices are preferred, but cell phone dictionaries are permitted
    - Students using cell phones are asked to sit on the back row on the side nearest the rear door of the room.  Other students are asked to sit elsewhere.
    - Students using paper dictionaries  or dedicated translation devices
    - may sit anywhere.
* All students, including both native and non-native speakers may ask questions about wording or vocabulary as long as the question does not directly answer a question.

I try to take my inspiration for test questions from "Check Point" questions from the book.  I may not use the questions directly, but they are most often the inspiration.

Chapter 6, Methods

|  |  |  |
| --- | --- | --- |
| Start  Page | Questions | Notes |
| 211 | 6.1 though 6.10 | 6.1: Method Benefits – reusability and segmentation  6.2 public static variable type name(variabletype passedvalue){}  6.3 set int result to 2 by default  6.4 True  6.5 void is the return type of main  6.6 Compile error. See previous. Yes  6.7 parameter: sent to methods are passed-by-value: The "type" of data that a method can receive  place holder  Argument: The values that are passed to a method  Method signature: method name and parameter list together  6.8 public static double salescom (double amount, double rate  public static void calander (int month, int year)  6.9 2: static **void** method…. Int n, **int** m){  4: int method2 can’t take 3.4  Switch lines 3 and 4.  6.10 public class Test{  Public static double method(double I, double j){  While(i<j){  j--;  }  Return j;  }  } |
| 214 | 6.11 through 6.13 | 6.11 “pass by value” they match up.  6.12 the variables in line 3 need to be rotated.  6.13 pass-by-value: value of an argument is passed to a parameter: 2; 2,4,8,16,32,64; |
| 221 | 6.15 and 6.17 | 6.15 method overloading – two methods have the same name but different parameter lists within one class. Java determines which to use based on method signature. No; Yes  6.16 methods overload each other.  6.17 a. First b. second c. first |
| 223 | 6.18 through 6.19 | 6.18 local variable – variable defined inside a method  6.19 the scope of a variable is the loop or method it is defined in |
| The book does not have any questions on it, but what is "stepwise refinement?"   * How would you use stepwise refinement in program development?  Divide the program into smaller sub-problems, which can be further divided into smaller tasks. One method at a time. * Would stepwise refinement make programming easier or harder?  Explain. Easier. It’s about working you’re way up from the simple to the hard stuff. * Would stepwise refinement help you write more reliable programs?  Explain. Yes, dividing into methods makes finding broken pieces easier to find and replace. * Would stepwise refinement help a team cooperate to write a complex program?  Explain. Yes, being able to divide the codes means dividing the workload, which is quicker development time. | | |

Chapter 7 Single Dimension Arrays

You should be comfortable doing standard array tasks such as

* Finding the minimum and maximum values in an array
* **public static double** findMIN(**double**[] list, **int** n) {  
   **double** minsofar = Double.***MAX\_VALUE***;  
   **if**(n<=0){  
   minsofar=0;  
   }**else**{  
   **for** (**int** i = 0; i < n; i++) {  
   **if** (list[i] < minsofar) {  
   minsofar = list[i];  
   }  
   }  
   }  
   **return** minsofar;  
  }  
    
  **public static double** findMAX(**double**[] list, **int** n) {  
   **double** maxsofar = Double.***MIN\_VALUE***;  
   **for** (**int** i = 0; i < n; i++) {  
   **if** (list[i] > maxsofar) {  
   maxsofar = list[i];  
   }  
   }  
   **return** maxsofar;  
  }
* Finding the average number in an array
* **public static double** calculateAvg(**double**[] list, **int** n) {  
   **double** sum = 0;  
   **double** average;  
   **for** (**int** i = 0; i < n; i++) {  
   sum += list[i];  
   }  
   average = sum / n;  
   **return** average;  
  }
* Linear searching an array

public static int linearSearch(int[]list, int key){

for(int i=0; i<list.length/\*n\*/; i++){

if(key==list[i]){

return i;

}

}}

* Using an array of counters

While (count[i]<100){

I think every programmer should be able to sort an array if they are stranded on a desert island and don't have a textbook.  But I won't ask you to write the complete sort on this exam.

* You should be able to modify the selection sort on page 270
  + Use n instead of .length: public static void SORT(String[ ]words, int n){…for(int i=0; i<n-1; i++){
  + If I give you the code with logic errors, you should be able to correct it.
  + Modify it to sort Strings using compareTo If(currentMin.compareTo(words[j]>0){
  + Modify it to sort in ascending or descending order. If(currentMax.compareTo(words[j]<0){
  + what would happen if the if statement in line 17 (and the corresponding } ) were omitted?  Would the sort still work?  Explain. No. That sets the sort to the start of the array.
  + What is the purpose of using a "final" integer constant to define the size of an array instead of a literal? Makes the Array more stable.

What is the "Growth Rate" of an algorithm.  Why do we care?  Know O(N), O(log(N)), and O(N2).   What is the growth rate of each of the following?

* Linear search = O(N)
* Binary search = O(log(N))
* ~~Insertion~~ Selection sort = O(N^2) == Exponential Growth Rate
* Constant Rate = O(1)
* Averaging an array = **public static double** calculateAvg(**double**[] list, **int** n) {  
   **double** sum = 0;  
   **double** average;  
   **for** (**int** i = 0; i < n; i++) {  
   sum += list[i];  
   }  
   average = sum / n;  
   **return** average;  
  }
* Find the largest or smallest item in an array = **public static double** findMIN(**double**[] list, **int** n) {  
   **double** minsofar = Double.***MAX\_VALUE***;  
   **if**(n<=0){  
   minsofar=0;  
   }**else**{  
   **for** (**int** i = 0; i < n; i++) {  
   **if** (list[i] < minsofar) {  
   minsofar = list[i];  
   }  
   }  
   }  
   **return** minsofar;  
  }  
    
  **public static double** findMAX(**double**[] list, **int** n) {  
   **double** maxsofar = Double.***MIN\_VALUE***;  
   **for** (**int** i = 0; i < n; i++) {  
   **if** (list[i] > maxsofar) {  
   maxsofar = list[i];  
   }  
   }  
   **return** maxsofar;  
  }
* Sum the elements in a 2 dimensional array = **for** (**int** i = 0; i < ROWS; i++) {  
   **for** (**int** j = 0; j < COLUMNS; j++){ sum += list[i][j];   
   } selection sort
* Sum the elements in a 3 dimensional array (we didn't talk about that, but think about it) O(N^3)

|  |  |  |
| --- | --- | --- |
| Start  Page | Questions | Notes |
| 251 | 7.1 through 7.11 | * What is "the heap" in Java * What is "garbage collection" in Java – auto-deletion of unneeded arrays. |
|  |  | 7.1 double **[]** arrayname; arrayname=new double[SIZE]  7.2 when size is defined in reference  7.3 x is 60 The size of numbers is 30.  7.4 Array elements same type: T  Array size is fixed when reference variable is declared: T  Array size is fixed after it is created: T  Array elements must be primitive: F  7.5 2, 5, and 6  7.6 through the index starting at [0]  7.7 ,a[2]  7.8 a. double[]arrayname=new double[10];  b. arrayname[LAST]={5.5};  c. .println( arrayname[0]+arrayname[1]);  d. **for** (**int** i = 0; i < n; i++) {  sum += list[i]; }  e. **double** minsofar = Double.***MAX\_VALUE***; **if**(n<=0){  minsofar=0; }**else**{  **for** (**int** i = 0; i < n; i++) {  **if** (list[i] < minsofar) {  minsofar = list[i];  }  } }  f. **for** (**int** i = 0; i < n; i++) { r[i]=Math.random\*100;}  g. double[] arrayname = {3.5, 5.5, 4.52, 5.6}  7.9 syntax error  7.10 3: double[] r= new double[100];  7.11 0 1 2 3 4 5 |
| 257 | 7.13 and 7.14 | * You do not need to know the "clone" method of copying arrays. * You do not have to know all 5 of the parameters of arraycopy, but you should know what they do. * Question 7.13 is important because it relates to a topic that is important to the rest of the course.  It relates to what is going on in the heap and with garbage collection. |
|  |  | 7.13 arraycopy(source, 0, t, 0, source.length);  7.14 No. It trashes the old array and makes a new one |
| 264 | 7.16 through 7.18 | 7.16 False. Arrays are passed by reference  7.17 number is 3 and number[0] is 3  7.18 The heap |
|  |  |  |
| 269 | 7.22 | * You should understand how a binary search works, but you don't need to be able to code it. * You only need to know the selection search. * Public static int binarySearch(int[]list, int key){   Int low=0;  Int high = list.length-1;  While (high>=low){  Int mid=(low+high)/2;  If(key<list[mid]){  High=mid-1;}  Else if(key==list[mid]){  Return mid;}  Else{  Low=mid+1;}  }return -1;} |
|  | Skip ~~7.24~~ through ~~7.28~~ | I don't want you using the Arrays class until I am convinced that the most of you are comfortable with manual manipulation of arrays.  In other words, after you don't need the Arrays class you will be able to use it. |
| * The book tends to use the .length property of arrays to control loops.   I wanted you to modify the listing on page 250 to use "n."   What is the difference?   When is it appropriate to use the "n" approach, and when is it appropriate to use the .length property? N takes Nulls into account when doing things like sort. | | |

You should be able to write methods to do common tasks with single dimensioned arrays.  In each case assume that there is a constant called MAX that indicates the physical size of the array, and a value n that represents the actual number of items in the array.  I used String types in the examples, but other types may be used.

* Read an array from a file:
  + Read an array from a file called "input.txt."
  + There is an unknown number of items in the file.  There may be more or fewer items than there are physical spots in the array
  + The method should fill the array and return n, the actual number of items in the array.

public static int readFile(String[] list{

Scanner input;

int n = 0;

try{

input=new Scanner(new File(“input.txt”));

while (input.hasNext()&& n<MAX){

list[n]=input.hasNext();

n++;

}

input.close();

}catch(FileNotFoundException e){

e.printStackTrace();

System.err.println(“ERRORS!”);

System.exit(1);

}

return n;}

* Print an array
  + Public static void print(String[] list, int n){

System.out.println(“------“);

For(int i=0; i<n; i++){

System.out.printf(‘\t%s\n”, list[i]);

}}

Strings

We only covered the String class as a set up for classes.  Later in the course we will swing back around and pick up some of the dusty corners of the String and StringBuffer class.

For now, know the following

* Why is "String" capitalized? It’s not a primitive value
* What is a reference variable?  What other reference variables have we seen in this course? is declared to be of a specific type and that type can never be changed. Reference variables can be declared as static variables, instance variables, method parameters, or local variables. Actually calls from another area instead of placing their. String. Boolean.
* What happens when a String is instantiated? A word is placed in the heap.
* Why can't you use the relational operators like <, >, or == with strings? Those are for primitives life int and double.
* How does compareTo work in practice?   How would you modify common searches and sorts of primitive data types to accommodate Strings? Hexidecimal value A.CompareTo(ValueB) > 0: A-B to get 1, -1, or 0. This example is basically looking to see if A is bigger than B

Classes

What is the difference between a Class and an Object?

Class – blueprint that defines what data items and verbs are available through abstraction

Object – an instance of a class