ICS 208 – Exam Review

## **Symbols, Punctuation – What is each used for?**

Some have more than one use.

|  |  |
| --- | --- |
| 1. { } | Store Arrays, open close methods; for loops; while loops. |
| 2. [ ] | Arrays, can use them to declare array, and access elements |
| 3. ( ) | Methods inclose parameter list, also used for method call. Used around Boolean expressions too. |
| 4. \ | Special character about to arrive |
| 5. / | Division |
| 6. // | Comments, sign code, explain, comment out code to make program work |
| 7. \\ | Two backslashes make one slash in an output line. |
| 8. \\* \*\ | Used to make multi-line comments |
| 9. + | Used to add, or add strings |
| 10. = | Assignment operator, changes value of variable, all types use this |
| 11. == | Used in expressions to test conditions, all variables except for String |
| 12. .compareTo | Used to compare two String variables |
| 13. .equals | Is the ‘=’ for strings |
| 14. && | Boolean Operator and both conditions have to be true for it to be true |
| 15. || | Boolean Operator and one condition has to be true for it to be true |
| 16. \* | Multiplication |
| 17. % | Gives remainder of number |
| 18. ! | Not Equals (reverse) |
| 19. ‘ – single quote | Goes around char values |
| 20. “ – double quote | Goes around string-values |
| 21. ; – semi-colon | Located at the end of statements |
| 22. : – colon | Used in switch or ? operators |
| 23. , – comma | Separate items in a list (array, parameters) |
| 24. . – period | Used to call a method (.setEnabled, .set) |
| 25. ? | Used for the ? operator |

# **Terms List – Why is each used? What is each one?**

## **Unit 1 – Binary, Input, Output**

|  |  |
| --- | --- |
| Assignment | Uses the assignment operator (=) to assign the result of an expression to a variable.  Ex. *int a = (b \*c) / 4;* |
| Class | The blueprint from which individual objects are created.  Ex. *class CLASS {}* |
| Comments | Useful to summarize code and make it easier to understand. Also it makes it easier to debug code by commenting out the unnecessary aspects. |
| Constructor | A bit of code that allows the user to create objects from a class.  Ex. *public static void main (String args []) {*  *new CLASSNAME ();* |
| Else | A statement that is executed only if the *if* condition is false. |
| If | A statement that is executed only if the condition is true. |
| Input | Any information that is needed by your to complete its execution.  Ex. *IBIO.input* |
| Integer Division | Division of integers. |
| Keyword | One of the 50 reserved words that have a predefined meaning in the language. These cannot be used for variables, methods, classes or as any other identifier.  Ex. *Boolean, string, char, double, int* |
| Main Method | An essential method needed in a class. |
| Math.PI | Pi function for Java. |
| Modulus | When a number is % by another number the result given is the remainder.  Ex. *10 % 3 = 1;* |
| Output | Any information that the program must convey to the user.  Ex. *System.out.print* |
| Type |  |
| Unique variable names | Names you give to variables.  Ex. *int num* |
| Variable | A variable provides the user with named storage that the program can manipulate. |
| Binary | Store data on comp, with just “1” and “0” |
| ASCII | The most common format for text files in computer and on the internet. |
| Unicode | An international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs. |
| Hexadecimal | Relating to or using a system of numerical notation that 16 rather than 10 as its base. |
| ASCII art (which isn’t related to ASCII) | Gives some visual effects and when using CLI (command line interface) |
| Flow chart | A visual presentation of a flow of data in a program. |
| Data Type | Type of data that can be stored in a variable.  Ex. *char, String, Boolean, int, double* |
| Smart Home | A home equipped with lighting, heating, and electronic devices that can be controlled remotely by phone or computer. |
| Google Car | A car that can drive without any human aid. |
| Analog | Anything that is non-digital. |
| Digital | Anything that is not analog. |
| Singularity | A hypothetical event in which artificial general intelligence would be capable of recursive self-improvement.  Year of hypothetical singularity (2045). |
| Ray Kurzweil | Stated that his *law of accelerating returns* which predicts an exponential increase in technologies like computers, genetics, nanotech, robotics and AI, will occur in the year 2045. |
| 3D Printing | A process of making a physical object from a three-dimensional digital model, typically by laying down many successive thin layers of a material. |
| Vernor Vinge | Computer scientists who thinks that in thirty years, mankind will “have the technological means to create superhuman intelligence.” ~ "The Coming Technological Singularity" (1993) by Vernor Vinge. |
| IFTTT recipe | IFTTT (If This Then That) is a web-based service that allows users to create chains of simple conditional statements, called “recipes”. |
| Superhuman AI (4 routes) | Vinge predicted four ways the singularity could occur:  1. The development of a computer that can think for themselves and are superhumanly intelligent.  2. Networks might become so intelligent that they might start thinking for themselves (such as Internet).  3. Computer/human interfaces become so intimate that user may become superhumanly intelligent.  4. Biological science may find ways to improve upon natural human intellect (cyborgs). |

## 

## **Unit 2 – Loops and Methods**

|  |  |
| --- | --- |
| AND | Boolean Operator and both conditions have to be true for it to be true. |
| Boolean | A binary variable that contains only two possibilities (true, false). |
| Boolean expression | An expression that evaluates to a value of Boolean data type: true or false.  Ex. *!((1 >= 4) && (2 <= 2)) = true;* |
| Global variable | A variable that is declared at the start of the program (outside the methods). |
| Initialize | When you declare a variable for the first time.  Ex. *double number = 2.4;* |
| Local variable | A variable declared inside a method. |
| Loop stopping condition | A statement that causes the program to either execute the loop or exit the loop. |
| Loop stopping variable | A variable that is initialized, which will be then used to test the loop stopping condition. |
| Method | A set of code which is referred to by name and can be called at any point in the a program simply by using the method’s method call. |
| Method Calls | A statement used to invoke a method.  Ex. *method (a, x);* |
| Method Signature | Method name and the number and type of its parameters, found in the first line of the method. |
| NOT | A Boolean operator that returns TRUE if its operand is FALSE, and vice versa. |
| OR | Returns the Boolean value true if either or both operands are true and returns false otherwise. |
| Parameter lists | A list of parameter declarations.  Ex. *public void method (int x, String y);* |
| Parameter Types | Parameter types include data types. |
| Parameters | Special kind of variable that is passed into the method. |
| Return Lines | Lines that return the value of a method.  Ex. *return x;* |
| Return Types | Return types include date types. |
| Trace |  |
| PDLC | Product Development Life Cycle. |
| Investigation | Research phase of the PDLC. Here, the problem is defined, and descriptions of the program’s inputs, processing, outputs, and user interface are written. |
| Design | Design phase of the PDLC. Developing a detailed logic plan, flowcharts, structure charts. |
| Code | Code phase of the PDLC. Translate the design into code. Testing of the program also takes place (Alpha, Beta, Gold Master) |
| Evaluation | Reflection phase of the PDLC. Review and revise internal documentation. |
| Steven Johnson | American popular science author and media theorist. |
| Adjacent Possible (Johnson) | “Captures both the limits and creative potential of change and innovation” ~Johnson  A shadow, hovering on the edges of the present state of things, a map of all the ways in which the present can reinvent itself.  **Definition Ms.Gorski gave us:**  From our current level of understanding, only certain discoveries are probable. We build on the knowledge we have - leap froggin forward is rare. The world has to be ready for the resulting product or service when it finally emerges. |
| Liquid Networks (Johnson) | Meetings where everybody got together and shared their kind of latest date and findings, often pointing out mistakes that could be avoided.  Where lots of different ideas that are together, different backgrounds, different interests, jostle with each other and this environment will lead to innovation.  **Definition Ms.Gorski gave us:**  Every ideas is fundamentally a network of ideas - our ideas build on other ideas and other people. |
| Slow Hunch (Johnson) | Notion that innovative thinking is a slow and gradual process rather than an instant moment of inspiration.  **Definition Ms.Gorski gave us:**  An innovative idea doesn’t happen in a vacuum nor in a spontaneous moment of clarity. Instead the idea evolve over the years. |
| Serendipity (Johnson) | The occurrence and development of events by chance in a happy or beneficial way.  **Definition Ms.Gorski gave us:**  The occurrence and development of events by chance in a happy or beneficial way. Great ideas occur by accident, when you are doing something else, maybe completely unrelated. |
| Error (Johnson) | **Definition Ms.Gorski gave us:**  You are working in one direction, make an error, and it leads you to another discovery. |
| Exaptation (Johnson) | **Definition Ms.Gorski gave us:**  Many discoveries get made by taking ideas from other disciplines and apply them to a new one. |
| Platforms (Johnson) | **Definition Ms.Gorski gave us:**  When you create an environment that allows a lot of new connections (a network), discoveries are possible. Coffee houses (1650), Salons (1700-1800), Social Media (now), Silicon Valley (now). |
| Structure Chart | a chart which shows the breakdown of a system to its lowest manageable levels. |

## 

## 

## **Unit 4 - Applets**

|  |  |
| --- | --- |
| Applet | a very small application, especially a utility program performing one or a few simple functions. |
| Libraries | a collection of precompiled routines that a program can use. |
| Imports | the automated or semi-automated input of data sets between different software applications. |
| GUI | GUI (Graphical User Interface) is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.  Basically, this has pictures. |
| CommandLine | The line on the display screen where a command is expected. Generally, the command line is the line that contains the most recently displayed *command prompt* .  A CLI (command line interface) is a user interface to a computer's operating system or an application in which the user responds to a visual prompt by typing in a command on a specified line, receives a response back from the system, and then enters another command, and so forth.  Basically this has no pictures. |
| accessor | Accessors use getters and setters  Ex. *.get and .set* |
| ActionCommand | The command string associated with the ActionEvent. |
| ActionEvent | A semantic event which indicates that a component-defined action occurred. |
| ActionListener | It listens and sees if any buttons on the screen are clicked. |
| ActionPerformed | Method where action command is sent into and a set action is performed depending on the ActionCommand |
| add to applet | Adds the widgets that are created to the applet so they can be displayed on the screen |
| construction | Creating an instance of a variable. (Declaring it). |
| createImageIcon | A method that lets users display images. |
| e.getActionCommand() | Gives you a String representing the action command. |
| For Loop | A for loop is an algorithm that allows users to repeat a piece of code over again as they desire. |
| init | A method containing all the necessary code to run the applet. |
| JButton | It is used to display a button on the applet. |
| JLabel | Display text and pictures on the applet. |
| Jtextfield | Displays a input box on the applet. |
| Light based colours | Magenta, Cyan  RGB (Red, Green, Blue) |
| mutator | When you give a new value to a already initialised variable. |
| Object | Programming entities that have basic characteristics: Identity, Type, State, Behaviour  Ex. public static void main(String []args){  class object = new class("object1");  } |
| Panel | A container class, where the user can add JLabels, JButtons, and JTextFields in a grid layout. |
| RGB colours | System of colours used on a computer display. |
| Swing | Swing is a set of program components for Java programmers that provide the ability to create graphical user interface ( GUI ) components. |
| While Loop | A loop that will repeat the code that it contains unless the stopping condition has been met. |
| Widget | A widget is a block of data that can be used in an applet.  Ex. *JLabel, JButton, JTextField* |
| Denise Melanson | Someone who had received a fatal dose of chemotherapy due to a system failure. |
| Screen Flow Charts | A chart that shows the flow of screens in an applet. |
| Visually Appealing (GUI design) | Use colour and pictures. Change the fonts. Resize the applet to one that displays the content well. Similar widgets should be the same size. |
| Clear Instructions (GUI design) | Provide clear titles. Put prompts in front of JTextfields. Have useful information on JButtons. Provide instruction in JLabels. |
| Restricts Input (GUI design) | Uses JButtons instead of JTextFields when possible - users are more likely to mistype then they are to misclick. Disables buttons when they should not be clicked. |
| Widget Arrangement (GUI design) | Like items are grouped with like items Widgets appear in the order the user will need them. This makes it easy to find what you need, Kike items should have similar formatting. |
| Error Handling (GUI design) | Have a reset/undo button to allow the user to start over if they make a mistake. Code if statements to handle invalid data entries. |

## **Unit 5: Arrays**

|  |  |
| --- | --- |
| 2D array | A two-dimensional array is really nothing more than an array of arrays (a three-dimensional array is an array of arrays of arrays). Think of your dinner. You could have a one-dimensional list of everything you eat: *(lettuce, tomatoes, steak, mashed potatoes, cake, ice cream)*  Or you could have a two-dimensional list of three courses, each containing two things you eat: *(lettuce, tomatoes) and (steak, mashed potatoes) and (cake, ice cream)*  In the case of an array, our old-fashioned one-dimensional array looks like this: *int[] myArray = {0,1,2,3};* And a two-dimensional array looks like this: *int[][] myArray = { {0,1,2,3}, {3,2,1,0}, {3,5,6,1}, {3,8,3,4} };* |
| Algorithm | a step by step set of operation to be performed. |
| Array | an arrangement of items in computer memory. |
| array type | Array data type, used in a programming language to specify a variable that can be indexed. |
| Bentley's example | Good hardware cannot be substituted for bad software. |
| Big-Oh notation | Used in Computer Science to describe the performance or complexity of an algorithm. |
| Binary search | Finds the position of a target value within a sorted array.  **Code:**  1 int[] data; 2 int size; 3 4 public boolean binarySearch(int key)  5 { 6 int low = 0; 7 int high = size - 1; 8  9 while(high >= low) { 10 int middle = (low + high) / 2; 11 if(data[middle] == key) { 12 return true; 13 } 14 if(data[middle] < key) { 15 low = middle + 1; 16 } 17 if(data[middle] > key) { 18 high = middle - 1; 19 } 20 } 21 return false; 22 } |
| Binsort | A sorting algorithm that works by distributing the elements of an array into a number of buckets.  It is used when the values in array are within a small range.  **Code:**  **public static void sort(int[] a, int maxVal) {  int [] bucket=new int[maxVal+1];    for (int i=0; i<bucket.length; i++) {  bucket[i]=0;  }    for (int i=0; i<a.length; i++) {  bucket[a[i]]++;  }    int outPos=0;  for (int i=0; i<bucket.length; i++) {  for (int j=0; j<bucket[i]; j++) {  a[outPos++]=i;  }  }  }** |
| Bubblesort | A simple sorting algorithm that repeatedly steps through the list to be sorted**,** compares each pair of adjacent items and swaps them if they are in the wrong order.  **Code:**  **public static void BubbleSort( int [ ] num )**  **{**  **int j;**  **boolean flag = true; // set flag to true to begin first pass**  **int temp; //holding variable**  **while ( flag )**  **{**  **flag= false; //set flag to false awaiting a possible swap**  **for( j=0; j < num.length -1; j++ )**  **{**  **if ( num[ j ] < num[j+1] ) // change to > for ascending sort**  **{**  **temp = num[ j ]; //swap elements**  **num[ j ] = num[ j+1 ];**  **num[ j+1 ] = temp;**  **flag = true; //shows a swap occurred**  **}**  **}**  **}**  **}** |
| Efficiency | The properties of an algorithm which relate to the amount of computational resources used by the algorithm. |
| element | A value in an array. |
| index | The location of an item in an *array*. |
| JButton array | An array of JButtons.  Ex. *JButton jbarray [] = new JButton [5];* |
| Linear search | A method for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is completed. The list does not need to be ordered.  **Code:**  **class LinearSearch  {  public static void main(String args[])  {  int c, n, search, array[];    Scanner in = new Scanner(System.in);  System.out.println("Enter number of elements");  n = in.nextInt();   array = new int[n];    System.out.println("Enter " + n + " integers");    for (c = 0; c < n; c++)  array[c] = in.nextInt();    System.out.println("Enter value to find");  search = in.nextInt();    for (c = 0; c < n; c++)  {  if (array[c] == search) */\* Searching element is present \*/*  {  System.out.println(search + " is present at location " + (c + 1) + ".");  break;  }  }  if (c == n) */\* Searching element is absent \*/*  System.out.println(search + " is not present in array.");  } }** |
| Maximum  & Minimum | Code to find the maximum and min.  *//array of 10 numbers*   1. **int** numbers[] = **new** **int**[]{32,43,53,54,32,65,63,98,43,23}; 3. *//assign first element of an array to largest and smallest* 4. **int** smallest = numbers[0]; 5. **int** largest = numbers[0]; 7. **for**(**int** i=1; i< numbers.length; i++) 8. { 9. **MAXIMUM:** **if**(numbers[i] > largest) 10. largest = numbers[i]; 11. **MINIMUM:** **else** **if** (numbers[i] < smallest) 12. smallest = numbers[i]; 14. } 16. System.out.println("Largest Number is : " + largest); 17. System.out.println("Smallest Number is : " + smallest); |
| Selectionsort | A combination of searching and sorting. During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position in the array. The number of times the sortpasses through the array is one less than the number of items in the array.  **Code:** **public static void SelectionSort ( int [ ] num )****{** **int i, j, first, temp;**  **for ( i = num.length - 1; i > 0; i - - )**  **{** **first = 0; //initialize to subscript of first element** **for(j = 1; j <= i; j ++) //locate smallest element between positions 1 and i.** **{** **if( num[ j ] < num[ first ] )**  **first = j;** **}** **temp = num[ first ]; //swap smallest found with element in position i.** **num[ first ] = num[ i ];** **num[ i ] = temp;**  **}** **}** |
| Sort | The arrangement of data in a prescribed sequence. |
| Search | An algorithm for finding an item with specified properties among a collection of items which are coded into a computer program, that look for clues to return what is wanted. |
| Sum/Average | **Code for Average (Sum is also coded):**  public class Average {   public static void **main**(String[] args) {   int count = 0; *// number input values*  double sum = 0.0; *// sum of input values*   *// read data and compute statistics*  while (!StdIn.**isEmpty**()) {  double value = StdIn.**readDouble**();  sum += value;  count++;  }   *// compute the average*  double average = sum / count;   *// print results*  StdOut.**println**("Average is " + average);  } } |
| .compareTo | The method compares the Number object that invoked the method to the argument. It is possible to compare Byte, Long, Integer, etc.  However, two different types cannot be compared, both the argument and the Number object invoking the method should be of same type. |
| .equals | Same thing as ‘==’ but for Strings. |
| Array Memory Diagram | Screen Shot 2016-01-22 at 7.41.14 PM.png |