CS200 Study Guide

By Aryan Krishna

Chapter 1: Programming Process

A computer *program* consists of instructions executing one at a time. A program consists of an input, a function that does something with the input, and an output.

- A ***program*** starts in main(), executing the statements within main's braces { }, one at a time.
- Each statement typically appears alone on a line and ends with a ***semicolon***, as English sentences end with a period.
- The int wage statement creates an integer variable named wage. The int wage = 20 statement assigns wage with 20.
- The print and println statements output various values.

The Scanner class is used to get inputs from the user:

- import java.util.Scanner; must be added top of the file
- Scanner scnr = new Scanner(System.in); in order to create scanner object

System.out.println()and System.out.print()to create outputs to terminal

Comments - // code for single line comment and /* code */ for multi line comments

Syntax errors are when the code written does not align with Java's rules and regulations. Java will give you an error message when you try to compile the code. The compiler will tell you what line and what error there is, so you can fix it.

Computers store all information in bits which are small binary switches that hold either the value 0 or 1.

Key parts of computer: Screen, Keyboard, Disk, RAM, Processor, Clock.

Programming is all about precision and that is why it is key to keep code clean(Indent, and give whitespace) where needed to increase code readability.

Chapter 2: Primitives and Expressions

Variables - have a data type, a name (item, x, y, name, etc), and are initialized/assigned using = some expression.

Declaration is decaring variable without assignment: int age;

Initializing is giving value to variable: age = 10; can also be done in 1 line: int age = 10;

4 types of arithmetic operators:

Addition (+), subtraction (-) multiplication (*), division (/). Regular Order of operation still applies.

Compound operators change a variable by a certain amount: +=,-=,/=,*=.

Int age = 8;

age*=4 is equivalent to age = age*4; so age would now be 32.

So far you have learned the data type int(integer). Doubles are another data type (all real numbers, including decimals.)

Use word final when declare variable it can never be changed: final int birthYear = 2004;

Integer division is how computer divides 2 ints. Just drop the decimal so 7/2 = 3 because 3.5-> 3. If either number is double then, regular division not integer division.

Modulo(%) is another operator, in other words, remainder. 7%2 = 1

Cannot divide by 0, otherwise will give you divide by zero error.

Type conversion is when a computer changes an int to a double or double to an int.

You can cast a variable to a different data type w the following syntax:

New Data type var name = (New Data type) old variable;

Another Data Type is a Character Literal aka char. Chars are 1 symbol variable.

Examples include ‘a’, ‘7’, ‘!’. Char firstLetter = “A”;

Some other numeric data types that store numbers in different ways with different levels of precision are : Long, Short, Byte, and Float.

Chapter 3: Using Objects and Defining Methods

A method is a set of instructions that we can create to do a repetitive task:

Method must be within a class public classname{}.

5 components of method header. public/private, static or non static(write nothing), void or has a data return type, the method name, and parameters/inputs.

One example could be public int getAge() { }

Methods can be called from within other methods. Void methods often print output

Main reason for methods is to improve code readability and efficiency

A ***module*** is a group of related packages. A ***package*** is a group of related classes. You can read API documentation to use public modules and packages in your code.

Strings:

- Collection of characters. String name = “Johnny”. Index starts at 0 so J is 0, o is 1, and y is 5.
- stringname.charAt(index) returns char at that index
- stringname.length() returns length of string

Variable names in java should be camel case: No spaces, first word lowercase, rest capitalized exp: theFirstName

Random Class:

- `Random randGen = new Random();`
- Many methods that can create random numbers with equal probability. Many uses

Debugging is an extremely useful tool, to find mistakes in your code efficiently.

Chapter 4: Branches

Boolean data type evaluates to True or False

If statement is in following syntax if(boolean statement){

*code that gets executed if boolean statement true}

If else and else:

If statement is in following syntax if(boolean statement){

*code that gets executed if boolean statement true}

Else if(boolean statement){

*code that gets executed if first statement false and second boolean statement true}

}

Else{

*code executed if all previous boolean statements false}

Can stack or burrow if else and else statements to create complex logic.

Compare numbers with ==: if(age==4){}. Compare strings name.equals("WIll"){}

Switch statements:

switch(a){

Case 0:

Case 1:

…

}

Executes code in case X that a is equivalent to

More String Operations:

indexOf(item) - returns index of item or -1 if not in string substring(index) returns string from index to end of string

substring(index1,index2) returns substring from index1 to index2 of original string

concat(string2) - adds string2 to the end of current string.

Strings are immutable so cannot be changed, only be fully reassigned.

Chapter 5: More Primitives, Objects, Branches, and Methods

Some char methods that return booleans are: isLetter(c), isDigit(c), and isWhitespace(c)

A conditional expression evaluates to true or false(a boolean value), and is what goes in the parentheses of an if statement

Doubles/floats are not stored with perfect accuracy so == does not properly check equivalence. To see if 2 nums are equal use: Math.abs(num1-num2)<Math.pow(10,-14);

&& is and operator evaluates to true if both conditionals true.

|| is or operator evaluates to true if either conditional true. Short circuiting is when the second conditional is not run bc its value will not change the overall value of the expression.

Each method call creates a new set of local variables, forming part of what is known as a ***stack frame***. A return causes those local variables to be discarded.

When a variable is declared in a method its scope is the method and can't be accessed outside of that method

Method overloading is creating 2 methods in 1 class that have the same name, but a different set of parameters. Therefore when the method is called based on what parameters are given, the computer knows which method to run.

Parameter error checking can prevent your program from crashing

Style Guidlines: https://google.github.io/styleguide/javaguide.html

Java Docs: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

A program's memory usage typically includes four different regions:

- ***Code*** — The region where the program instructions are stored.
- ***Static memory*** — The region where static fields are allocated. The name "static" comes from these variables not changing (static means not changing); they are allocated once and last for the duration of a program's execution, their addresses staying the same.
- ***The stack*** — The region where a method's local variables are allocated during a method call. A method call adds local variables to the stack, and a return removes them, like adding and removing dishes from a pile; hence the term

"stack." Because this memory is automatically allocated and deallocated, it is also called ***automatic memory***.

- ***The heap*** — The region where the "new" operator allocates memory for objects. The region is also called ***free store***.

Swapping 2 variables is extremely useful and is done with a temp 3rd variable:

```
Int a = 8; int b = 6;

Int temp=a;

Int a=b;

Int b=a;
```

Chapter 6: Loops

A ***loop*** is a program construct that repeatedly executes the loop's statements (known as the ***loop body***) while the loop's expression is true; when the expression is false, execution proceeds past the loop. Each time through a loop's statements is called an ***iteration***.

While loop performs iterations as long as conditional evaluates to true

while(conditional){

*code to be run}

Something in the code should eventually turn the conditional false to end the loop

For loops run a set amount of iterations in the following format:

for(int a = some constant; some conditional involving a; incrementing a){

} the following example will run the loop 10 times:

for(int a=0;a,10;a++){

}Nested loops are loops inside loops and allow more complex data structure analysis

break and continue statements can be used inside loops. Continue skips the rest of the code in the current iteration and break skips the rest of the loop entirely.

Some variables need to store a small set of named values. Enums can be used for this:

```
public enum LightState {RED, GREEN, YELLOW, DONE}
```

Do while loops: do{

}while(conditional)

Does one extra iteration of the loop than regular while loop.

Loops can be used for many important tasks: finding min/max of list, sorting lists, searching through letters of strings, and much much more.

Chapter 7: Arrays

Once a variable's scope is completed the memory of that variable is reallocated by the garbage collector.

An array is a data type that stores several pieces of data of same type.

Array declaration syntax: `dataType[] arrayName = new dataType[numElements];`
Arrays indexes range from 0 to arr.length-1
Setting array data: arr[index] = value;
Getting array data datatype data = arr[index];
Can loop through arrays to find min/max, sort, or do other things aswell.
A ***perfect size array*** is an array where the number of elements is exactly equal to the memory allocated
An ***oversize array*** is an array where the number of elements used is less than or equal to the memory allocated. Since the number of elements used in an oversize array is usually less than the array's length, a separate integer variable is used to keep track of how many array elements are currently used.

Chapter 8: More Arrays and Methods:

An array can be declared with two dimensions. `int[][] myArray = new int[R][C];` r represents rows and c the columns

All the same functions as 1D arrays apply so arr[2][3] gets row 2 columns 3 value.

An enhanced for loop goes through and list datatype including arrays in the format:

(Datatype varName : listName){

}

One common array modification is reversing an array:

```
for (i = 0; i < (userVals.length / 2); ++i) {

         tempVal = userVals[i];                                  // Temp for swap

         userVals[i] = userVals[userVals.length - 1 - i]; // First part of
swap

         userVals[userVals.length - 1 - i] = tempVal;      // Swap complete

      }
```

Unit testing is testing each part of code separately to see where the “bug” is.

An easy way to tell if a method needs a perfect size array or an oversize array is if there is a size parameter - that would indicate it is an oversize array.

All objects including arrays, unlike primitives, are passed by reference rather than value, so when an array is passed into a method as a parameter, it itself is changed. This is an important topic. Click here to learn more.

Methods using Arrays as parameters and returning arrays can be tricky. It is important to double check that the function does what you want it to do.

Chapter 9: Objects and ArrayLists

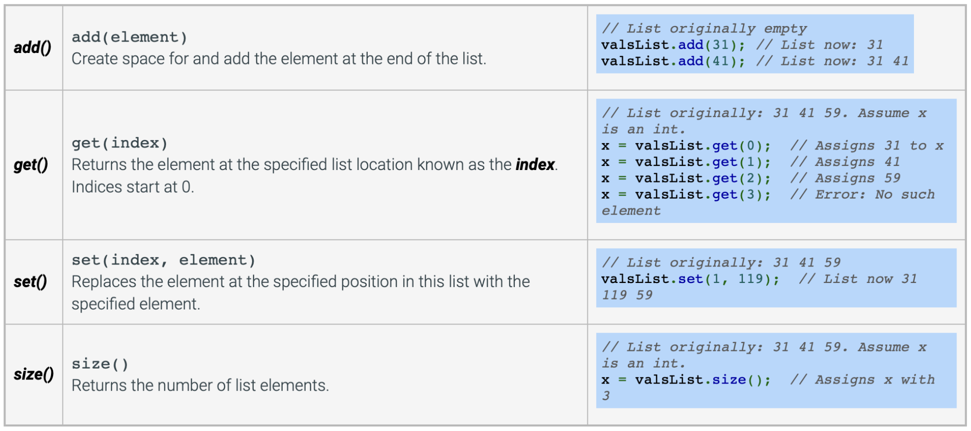
An *object* is a grouping of data (variables) and operations that can be performed on that data (methods).

A reference is a call to the memory location where the object is stored. Using the new keyword creates a reference to a new object: ObjectType varName = new ObjectType();

Wrapper classes take a primitive data type and put it into an immutable(unchangeable) object class. One example is the Integer which wraps the primitive int datatype. Primitive -> Wrapper is called autoboxing and Wrapper -> primitive is unboxing.

Arraylist is similar to array, in the fact that it is a list of 1 datatype, but where it differs is the fact that it's size can increase and decrease.

Initializing: `ArrayList<DataType> vals = new ArrayList<DataTyepe>();`



add()	`add(element)`
	Create space for and add the element at the end of the list.
		`// List originally empty`
		`valsList.add(31); // List now: 31`
		`valsList.add(41); // List now: 31 41`

get()	`get(index)`
	Returns the element at the specified list location known as the ***index***. Indices start at 0.
		`// List originally: 31 41 59. Assume x is an int.`
		`x = valsList.get(0);  // Assigns 31 to x`
		`x = valsList.get(1);  // Assigns 41`
		`x = valsList.get(2);  // Assigns 59`
		`x = valsList.get(3);  // Error: No such element`

set()	`set(index, element)`
	Replaces the element at the specified position in this list with the specified element.
		`// List originally: 31 41 59`
		`valsList.set(1, 119);  // List now 31 119 59`

size()	`size()`
	Returns the number of list elements.
		`// List originally: 31 41 59. Assume x is an int.`
		`x = valsList.size();  // Assigns x with 3`

One problem with arraylists is the efficiency. If you add a value to the middle of the arraylist, it has to shift every element to the right of it down one memory spot which takes time.

Chapter 10: Exceptions

An ***exception*** is an unexpected incident that stops the normal execution of a program. Ex: Dividing by zero or getting invalid input results in an exception. A program that does not handle an exception ends execution.

Programmers can intentionally throw an exception if a certain error or condition is met:

```
throw new Exception("Invalid area");
```

Methods can also have a throws at the end of the signature which means the method must throw that exception somewhere in the program:

```
public static double getArea(Scanner scnr) throws Exception {}
```

Can use try catch blocks to catch exceptions.

Try{

*code}

catch(Exception exp){

*code to be run if exception was thrown

} ***Command-line arguments*** are values entered by a user when running a program from a command line. A *command line* exists in some program execution environments, wherein a user types a program's name and any arguments at a command prompt. To access those arguments, main() can be defined with a special parameter args, as shown below. The program prints provided command-line arguments.

You can use said arguments in your main method, by calling upon the args parameter array

Chapter 11: File Input/Output

It is important for programs to be able to receive input from an user. 2 ways to do this in java are using the scanner class and using the System.in object.

When using the System.in object to receive user input it is important to use a throws IOException or else the program won't compile.

printf() can be used to print conjugated strings without the use of the addition operator. printf() uses symbols to represent datatypes, and then you must inout the parameters in the correct order to match the symbols and then the function prints said message w the imputed parameters.

```
System.out.printf("The %s account saved you $%f over %d years\n",

    account, total, years);
```

Format specifier	Data type(s)	Notes
%c	char	Prints a single Unicode character
%d	int, long, short	Prints a decimal integer value.
%o	int, long, short	Prints an octal integer value.
%h	int, char, long, short	Prints a hexadecimal integer value.
%f	float, double	Prints a floating-point value.
%e	float, double	Prints a floating-point value in scientific notation.
%s	String	Prints the characters in a String variable or literal.
%%		Prints the "%" character.
%n		Prints the platform-specific new-line character.

A programmer can read data from a string instead of from the keyboard (standard input) by associating a Scanner object with a String. Use a string instead of Scanner.in as parameter in Scanner object.

Scanner.next() and Scanner.nextInt() can help you parse through a string.

On the other hand an ***output string stream*** is a stream that can write to a String instead of to standard output. standard output).

Opening and reading from a file: The statement `fileByteStream = new FileInputStream(str);` creates a file input stream and opens the file denoted by a String variable, str, for reading. There are several methods that can be used to read from a text file such as: hasNext(), hasNextInt, next(),nextInt(), and more.

These are important for opening and writing to a file:

Action	Sample code
Open the file helloWorld.txt for writing	`FileOutputStream fileStream = new FileOutputStream("helloWorld.txt");`
Create a PrintWriter to write to the file	`PrintWriter outFS = new PrintWriter(fileStream);`
Write the string "Hello World!" to the file	`outFS.println("Hello World!");`
Close the file after writing all desired data	`outFS.close();`

When working with files, there are many places one could go wrong: Trying to open a file that doesn't exist, reading a number from a string that doesn't have any ints left, etc. Therefore, it is crucial to use exception handling to prevent your program from crashing.

EXP:
```
try {
         fileInStream = new FileInputStream(fileName);
         fileScanner = new Scanner(fileInStream);
         // Read file with fileScanner ...
         System.out.println("Average: " + dataAvg);
      }
      catch (FileNotFoundException e) {
         System.out.println("Cannot find " + fileName);
      }
```
Additionally it is important to close the file once you are done reading/writing to it with the .close() function

Chapter 12: Basic Classes

A class' ***public member methods*** indicate all operations a class user can perform on the object.

The ***new*** operator explicitly allocates an ***object*** of the specified class type. Ex: `Restaurant favLunchPlace = new Restaurant();` creates a Restaurant object named favLunchPlace.

Components of a class are: Private instance variables, constructors, and methods

Methods can be categorized into 4 categories: Getter, setter, helper, and main methods

Common practice is to declare private instance variables at the top of the class, and initialized in the constructor

Main method syntax:

```
public static void main(String [] args) {

   }
```

Just like methods, constructors can also be overloaded, with different sets of parameters. This is important because you may want different levels of customization at different points in your application.

The implicit “this” parameter allows you to call on the current object and its variables, and methods within the class itself