**-Discount:** Lets say you have 10 as discount percent. How do you take %10 off of price?

net price = price \* (100 - discountPercent) / 100)

**-Growth rate:** When you multiply a number by 1.00 it stays same. So growth is %0. When you multiply a number by 2.00 its size becomes twice the original. So the two digits right of decimal point shows growth in percentege. Multiplying by 1.65 will grow the number by %65. For growth over %100 you need to start using 2.xx, 3.xx etc. Multiplying by 3.00 means an increase of %200.

**-Error, change and difference:** [Relative change and difference](https://en.wikipedia.org/wiki/Relative_change_and_difference) (tag: numerical methods)

% = [(value after change/value before change) - 1] X 100

4 / 2.66 = 0.66. This means 2.66 is 66% of 4. This means 2.66 is what you get when you decrease 4 by 33%. (exercise 5.20)

difference = Math.abs((pi / lastPi -1) \* 100);

**-Palindrome:** To see if a number or string is palindrome or not dont waste time getting characters from both sides one by one and comparing them. Just use a temporary variable, reverse one of them and compare.

**­-Gathering/getting rid of certain digits:** We can use x % y and x / y to get rid of/gather certain digits. We use remainder operator to gather n digits from right side. We use division operator to gather n digits from left side.

1234 / 1000 gets rid of 3 digits to the right. Takes the left most digit. 1. Digit.

(1234 / 100 ) % 10 gets rid of 2 digits to the right and then takes the right most digit. 2. Digit.

(1234 / 10 ) % 10 gets rid of 1 digit to the right and takes the right most digit. 3. Digit.

1234 % 10 takes the right most digit. 4. Digit.

In order to get the 2 digits after decimal point, you can multiply by 100, get the last 2 digits. (exercise 5.18)

**-Rounding specific decimal places**

y = Math.floor(x + 0.5); // Rounding to integer

y = Math.floor(x \* 10 + 0.5) / 10; // Rounding to tenths

y = Math.floor(x \* 100 + 0.5) / 100; // Rounding to hundredths

y = Math.floor(x \* 1000 + 0.5) / 1000; // Rounding to thousandths

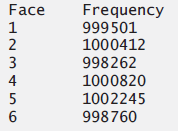
-If you divide or multiplicate a number with its base, you will push numbers right or left respectively.

**-Printing Tables**

-Left justified

System.out.println("Face\tFrequency"); // output headers

System.out.printf("1\t%d%n2\t%d%n3\t%d%n4\t%d%n5\t%d%n6\t%d%n", frequency1,   
 frequency2, frequency3, frequency4, frequency5, frequency6);



-Right Justified:You need to know the length of the strings in header other than last one. In this example we need to know the length of “Year” which is 4. So while we are printing we allocate 4 slots to first value to be inputted and then we put next value in the next 20 slot just like we did in header.

System.out.printf("%s%20s%n", "Year", "Amount on deposit"); // Header

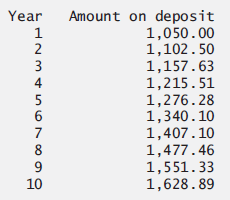
for (int year = 1; year <= 10; ++year)

{

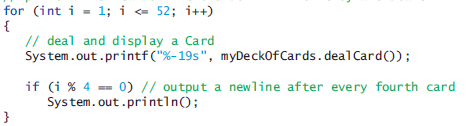
amount = principal \* Math.pow(1.0 + rate, year);

System.out.printf("%4d%,20.2f%n", year, amount);

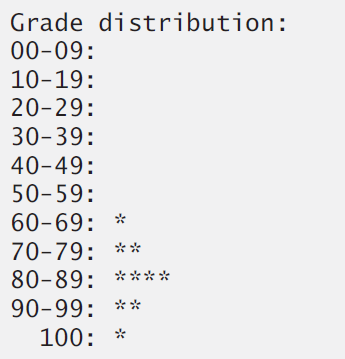
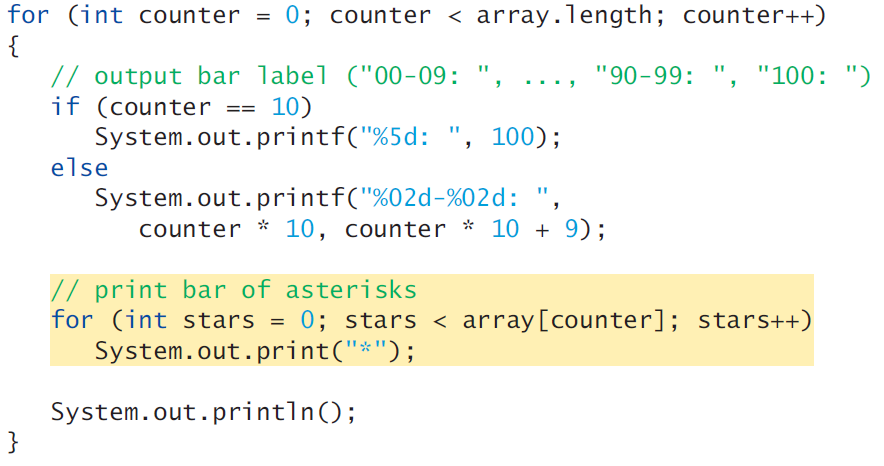
}



-Equal room for elements: When you want equal room for elements being printed horizontal just give them say 20 characters of room so all elements are printed in their own room.



Interval/Range: 00-09:, 10-19:, …, 90-99:, 100:



**-x and y coordinates on a two dimensional array:** When you are navigating through a two dimensional array, dont use array[positionX][positionY]. Because in computers first dimension controls row(vertical), and second dimension controls column(horizontal). Check exercise 7.21.

**-Prime numbers:** You dont have to check all numbers up to n/2. Checking all numbers up to sqrt(n) is enough.

You also dont need to check against all numbers from 1 to sqrt(n). You only need to check against prime numbers between 1 and sqrt(n). Using sieve of eratosthenes uses this fact to forward eliminate numbers. (C++ chapter 4 execises)

**-Greatest Common Divisor:** Use euclidian algorithm.

public static int gcd(int number1, int number2)

{

if (number2 == 0)  
 return number1;

else

return gcd(number2, number1 % number2);

}

**-Sum of squares:** (2 \* n + 1) \* (n + 1) \* n / 6;

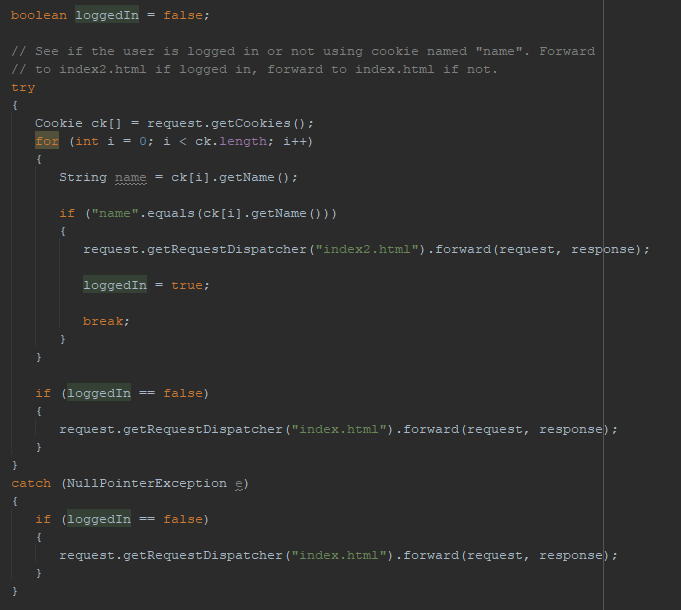
**-Complex nested for statements:** Here is how you should think while creating complex nested for statements. Lets say we have a one dimensional array and if multiple arrays have the same value the elements other than first one should be set to -1.

The first for statement should run one time for every element in our array. So it is easy to

When writing the second for statement you should first think about the first iteration. Which means the initial value of the counter of the second for statement. In our example the first iteration of the first for statement should compare array[0] with array[1]. Counter1 will be 0 so initial value of counter2 should be counter1 + 1. Initial value of counter2 should be counter1 + 1 in all iterations of first for statement because we start by comparing counter1 with counter1 + 1 and then we compare counter1 with counter1 + 2 and so on until we reach the last element.

Counter2 should keep incrementing as long as it is lower than length of array. So every iteration of outer for statement the inner for statement will execute one less time. We achieve this by using counter1 dynamic value in our counter2 initialization. It wont execute at all for last element because in the inner for statement the initial value of the counter2 will not be valid for loop continuation condition. So our loop contnuation condition must be counter2 < array.length.

**-Want to use “if else” but cant:** In the code below we look for the login cookie. We want to say “if you find the login cookie, let give the user the logged in website version. Else, give the user not logged in vebsite version. But since we are using a for loop to see if the login cookie exist, we have to either do the for loop, assign result to a boolean variable and then do an “if else” or we can just use a marker like we did in code below. But using if else is a better structured programming practice. Makes more sense and easier to understand. You want to make your program as easy to udnerstand as possible, you dont want to make it a riddle/puzzle.

****

**-Conversions**

**-Decimal to non decimal**

private static int convertToBase(int decimalNumber, int base) {

int result = 0;

int multiplier = 1;

while(decimalNumber > 0) {

int residue = decimalNumber % base;

decimalNumber = decimalNumber / base;

result = result + residue \* multiplier;

multiplier = multiplier \* 10;

}

return result;

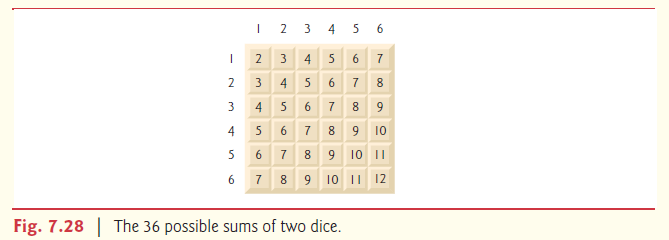
}

**-Universal time to standart time:**

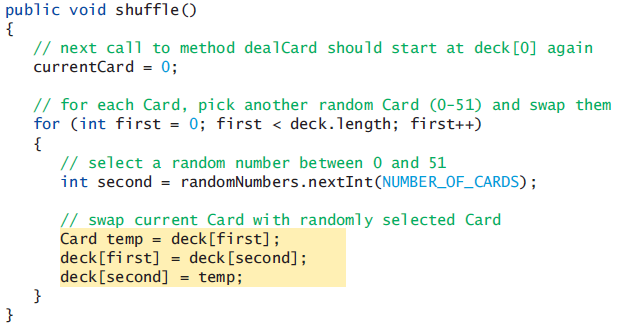
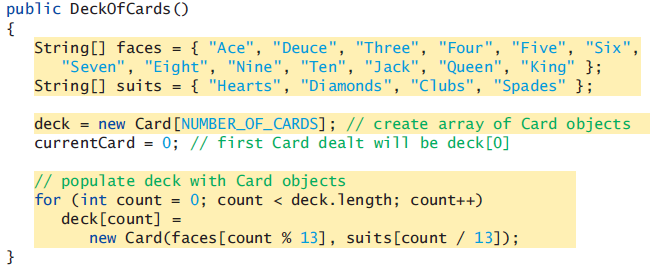
return String.format("%d:%02d:%02d %s", ((hour == 0 || hour == 12) ? 12 : hour % 12),

minute, second, (hour < 12 ? "AM" : "PM"));

**-Dice sample space:**



**-Creating and randomizing a card deck:** Using all avaiable suits and faces without missing any. And then randomizing them.



**-null vs empty string:** email == sees if the email object is pointing to something or not. Email.equals(“”) sees if the object email is pointing to is an empty string or not. If you didnt have the first check, the second check could have returned a null point exception. Also when you are using the equals method make sure to put the item you know for sure is a string to left to not get null point exception.

if (email == null || email.equals(""))

**-Date , Calculate age:**



(tag: java.time.Period, java.time.LocalDate)

**-Turn left, turn right, move forward**

C:\Computer Engineering\8- Java\4- Programs\- Book Exercises\Chapter 7\2- Exercises\7.21

<https://gamedev.stackexchange.com/questions/36046/how-do-i-make-an-entity-move-in-a-direction>

**-Using equations instead of many iterations:** Sum the odd integers between 1 and 99, using a for statement. Assume that the integer variables sum and count have been declared. (tag: sum)