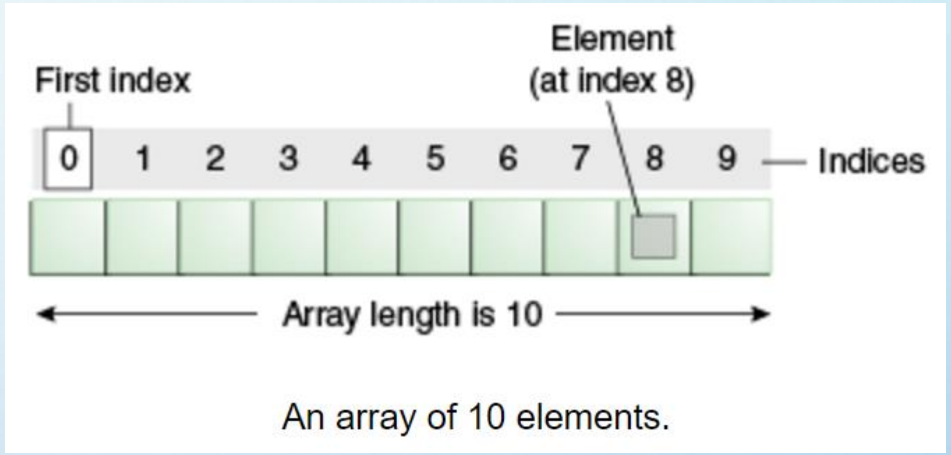
CIT 149
INF 120
FINAL EXAM REVIEW

ARRAYS - BASICS

- An ***array*** is a container object that holds a fixed number of values of a single type.
- An array is given a name
- The length of an array is established when the array is created.
- After creation, its length is fixed.
- Each item in an array is called an *element*
- Each element is accessed by its numerical *index (or subscript)*.

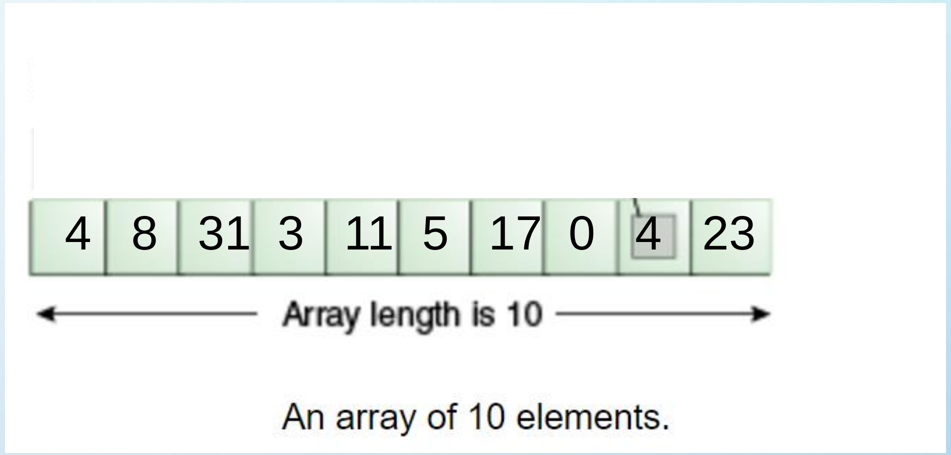


An array of 10 elements.

```
int[ ] age  = new int[10];
```

ARRAYS - BASICS

- What is stored in
  - age[8] - 4
  - age[1] - 8
  - age[10] – Subscript out of range



An array of 10 elements.

- Write a loop to display all elements in the array –

```
 for (int i = 0; i < age.length; i++)
{     System.out.println(age[i]);      }
```

ARRAYS – 2D

- Sample initialization with values supplied

int CandyBars [][]={ {5,10,6}, {6, 12, 16}, {9,15,8}, {10,8,18}, {7,7,10}, {18,2,9} };

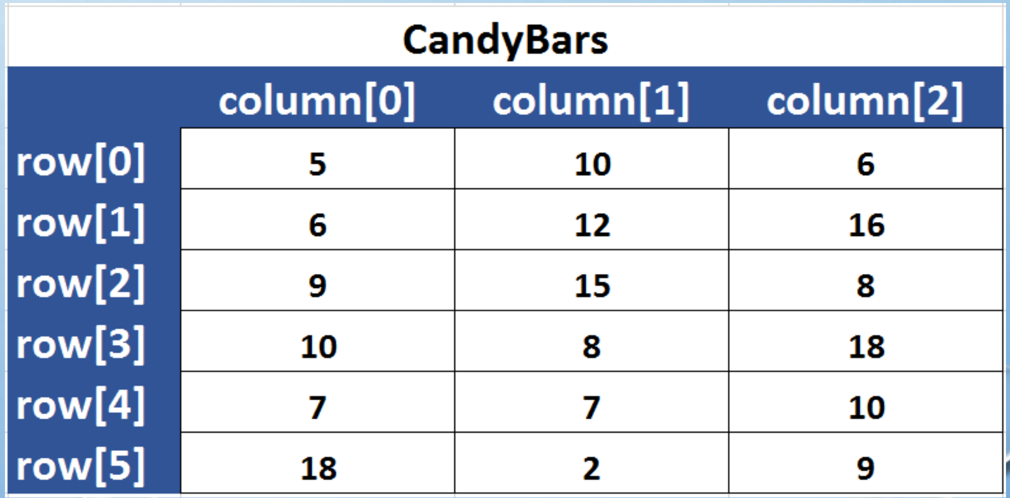
- Sample initialization without values

int CandyBars[][] = new int[6][3];

- What value is stored in

  CandyBars[4][2]?

  10



CandyBars

	column[0]	column[1]	column[2]
row[0]	5	10	6
row[1]	6	12	16
row[2]	9	15	8
row[3]	10	8	18
row[4]	7	7	10
row[5]	18	2	9

ARRAYS – 2D

- Write code to display the array in row format

```
for (int row = 0; row < CandyBars.length; row++)
  {     for (int column = 0; column < CandyBars[0].length; column++)
        {        System.out.print(CandyBars[row][column] + "\t");
        }
        System.out.println();
  }
```

CandyBars

	column[0]	column[1]	column[2]
row[0]	5	10	6
row[1]	6	12	16
row[2]	9	15	8
row[3]	10	8	18
row[4]	7	7	10
row[5]	18	2	9

CLASS REVIEW

- UML Diagrams
- Classes
- Objects
- Methods
- Instance data/variables
- Encapsulation
- Instantiation
- Constructor
- Getter (accessor)
- Setter (mutator)
- toString method
- Additional methods

GUI REVIEW

- Review Chapter 6 readings for multiple-choice questions
- There will not be any coding questions from Chapter 6 on the exam

String methods - Page 81 in Lewis

- charAt()
- compareTo()
- concat() or + operator
- equals() – used to compare string contents (not ==)
- replace()
- substring()
- toLowerCase()
- toUpperCase()
- contains() – not in book
- length property

Math methods - Page 90 in Lewis

- abs(): Math.abs(x) – absolute value
- ceil(): Math.ceil(x) – smallest whole # >= x
- floor(): Math.floor(x) – largest whole # <= x
- pow(): Math.pow(x,y) – x^y
- sqrt(): Math.sqrt(x) – square root of x

REVIEW BASICS FROM CHAPTERS 1-7

- Which operator instantiates an new object?

  = new

- What is the purpose of import statements?

  To have availability to the classes a program uses

- What are the increment and decrement operators?

  ++ and --

REVIEW BASICS FROM CHAPTERS 1-7

- Suppose x = 5 what is x after x++;

  6

- Suppose x = 5 what are the values of x and y after:

  y = x++;

  y = 5 and x = 6

- Suppose x = 2 and y = 3 what is the value of answer when answer = x * y++;

  6 (y is assigned to 4 after the assignment)

REVIEW BASICS FROM CHAPTERS 1-7

- Suppose x = 5 what is x after ++x;

  6

- Suppose x = 5 what are the values of x and y after:

  y = ++x;

  y = 6 and x = 6

- Suppose x = 2 and y = 3 what is the value of answer when answer = x * ++y;

  8 (y becomes 4 before the multiplication operation)

REVIEW BASICS FROM CHAPTERS 1-7

- What is displayed: System.out.println(“Answer: “ + 10 + 10);

  Answer: 1010

- What is displayed: System.out.println(“Answer: “ + (10 + 10))

  Answer: 20

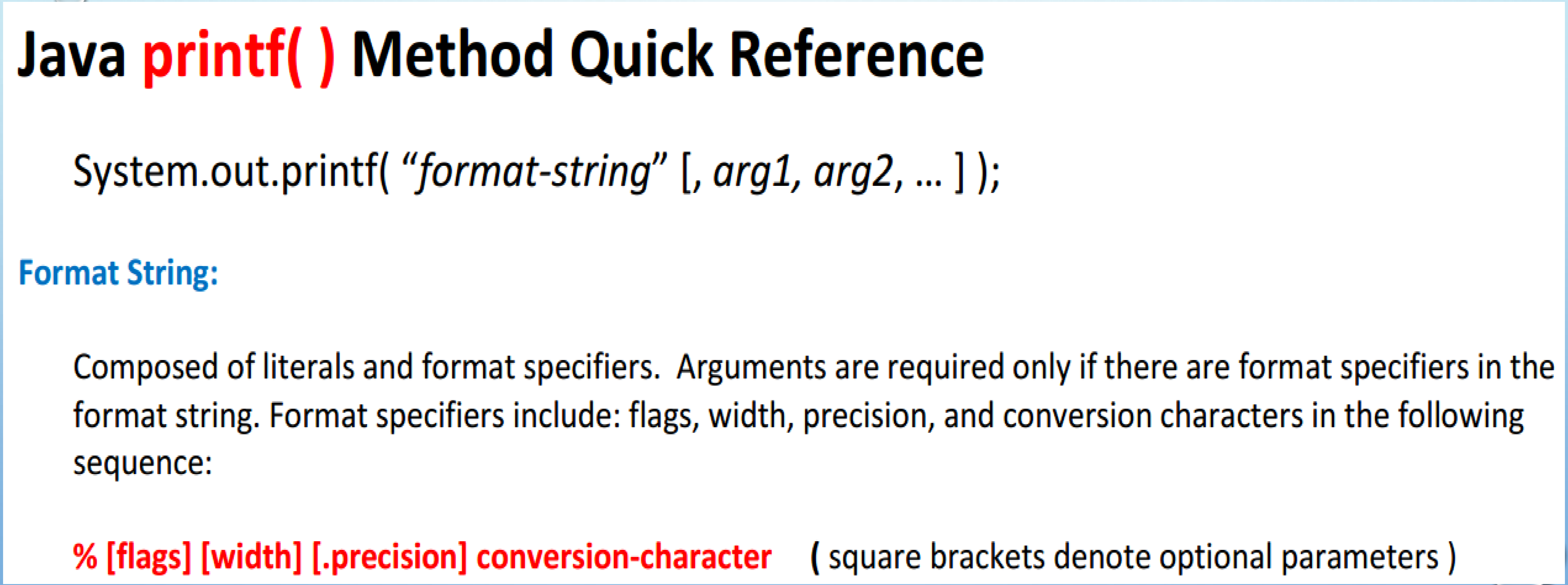
REVIEW BASICS FROM CHAPTERS 1-7

- Comments
- Enumerated lists – page 97
- Scanner class
  - Reading from keyboard
  - Reading from file
- Selection structures
  - if
  - switch
- Repetition structures
  - do
  - while
  - for
- Integer division
- Primitive data types
  - Amount of memory used by each

REVIEW EXTERNAL MATERIAL

- Clean Code Reading
- Towson Security Injections
- printf() and format()

USING PRINTF() AND FORMAT()



Java printf( ) Method Quick Reference

System.out.printf("*format-string*" [, *arg1*, *arg2*, ...]);

Format String:

Composed of literals and format specifiers. Arguments are required only if there are format specifiers in the format string. Format specifiers include: flags, width, precision, and conversion characters in the following sequence:

% [flags] [width] [.precision] conversion-character (square brackets denote optional parameters)

FORMATTING IN COLUMNS

% [flags] [width] [.precision] conversion-character (square brackets denote optional parameters)

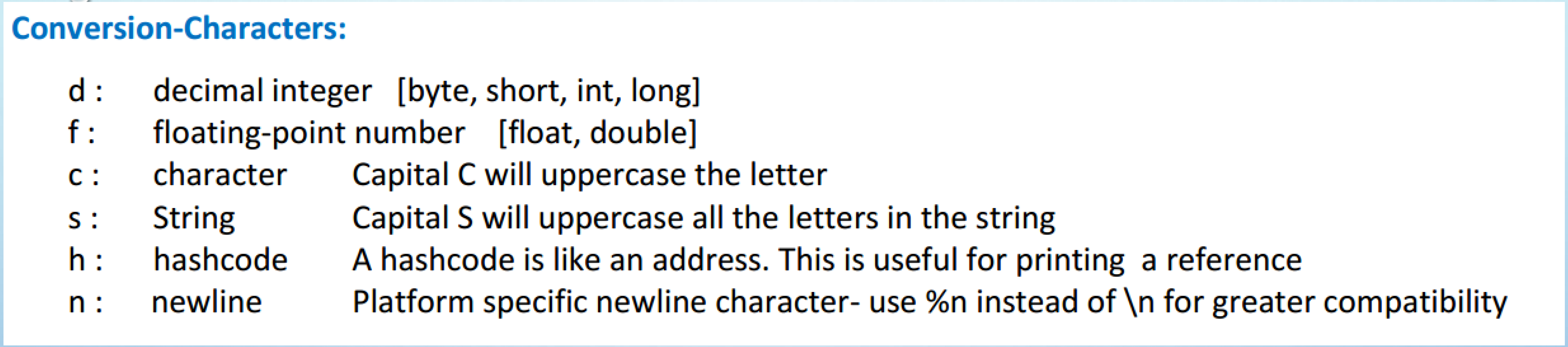
Flags:

-	:	left-justify (default is to right-justify)
+	:	output a plus (+) or minus (-) sign for a numerical value
0	:	forces numerical values to be zero-padded (default is blank padding)
,	:	comma grouping separator (for numbers > 1000)
	:	space will display a minus sign if the number is negative or a space if it is positive

Width:

Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output. Include space for expected commas and a decimal point in the determination of the width for numerical values.

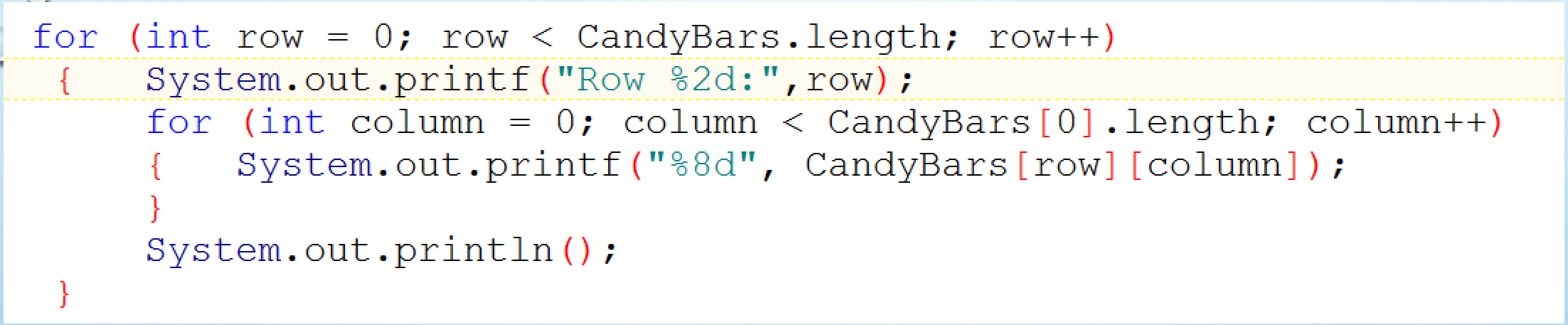
FORMATTING IN COLUMNS



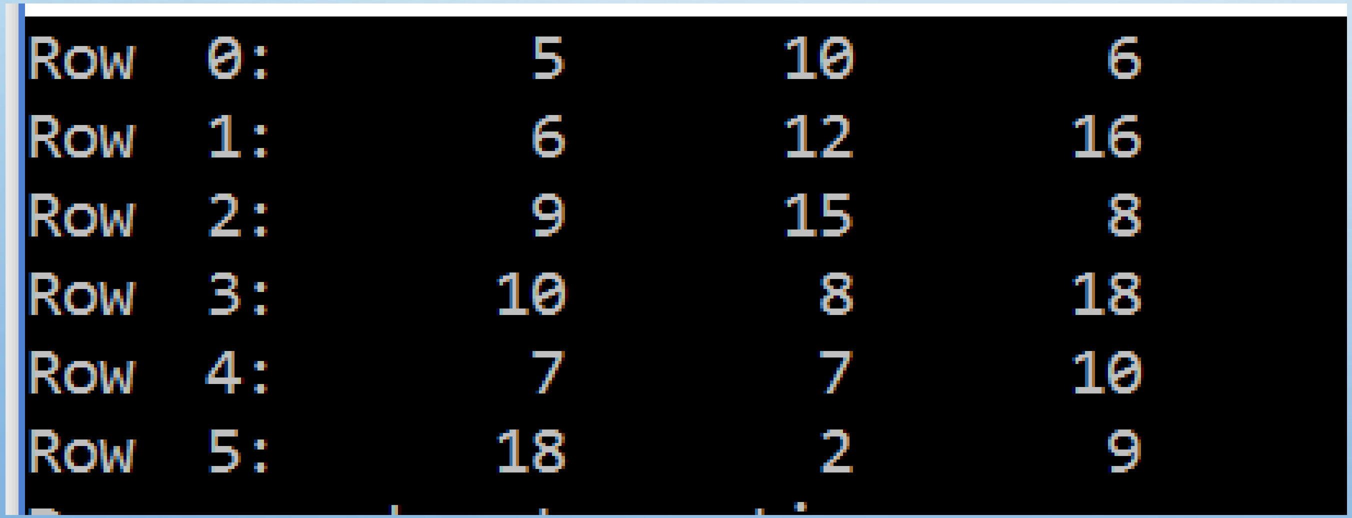
Conversion-Characters:

d :	decimal integer [byte, short, int, long]	
f :	floating-point number [float, double]	
c :	character	Capital C will uppercase the letter
s :	String	Capital S will uppercase all the letters in the string
h :	hashcode	A hashcode is like an address. This is useful for printing a reference
n :	newline	Platform specific newline character- use %n instead of \n for greater compatibility

FORMATTING IN COLUMNS



```java
for (int row = 0; row < CandyBars.length; row++)
 {    System.out.printf("Row %2d:",row);
      for (int column = 0; column < CandyBars[0].length; column++)
      {    System.out.printf("%8d", CandyBars[row][column]);
      }
      System.out.println();
 }
```



```
Row  0:       5      10       6
Row  1:       6      12      16
Row  2:       9      15       8
Row  3:      10       8      18
Row  4:       7       7      10
Row  5:      18       2       9
```