Indiana University
CSCI-C212: Introduction to Software Systems (F25) — Problem Set 2
Joshua Crotts — September 3, 2025

Strings and Conditionals

Important Dates:

- Assigned: September 3, 2025
- Deadline: September 10, 2025 at 11:59 PM EST

Objectives:

- Students work with certain string manipulation methods.
- Students begin to understand the different types of conditional statements and how they redirect program control flow.

What To Do:

For each of the following problems, create a class named `ProblemX`, where `X` is the problem number. E.g., the class for problem 1 should be `Problem1.java`. Write (JUnit) tests for each method that you design in corresponding test files named `ProblemXTest`, where `X` is the problem number. Additionally, write Javadoc comments explaining the purpose of the method, its parameters, and return value. **Do not round your solutions!**

What You Cannot Use:

You cannot use any content beyond Chapter 2.1. This includes recursion, loops, arrays, regular expressions, data structures, and so forth. Please contact a staff member if you are unsure about something you should or should not use.

Any use of anything in the above-listed forbidden categories will result in a **zero** (0) on the problem set.

Problem 1:

Design the `String cutUsername(String email)` method that receives an email address of the form `X@Y.Z` and returns the username. The username of an email address is `X`.

Problem 2:

Design the `boolean lessThan20(int x, int y, int z)` method that, when given three integers x, y, and z, returns whether or not one of them is less than twenty away from another. For example, `lessThan20(19, 2, 412)` returns true because 2 is less than 20 away from 19. Another example is `lessThan20(999, 888, 777)`, which returns false because none of the numbers have a difference less than twenty.

Problem 3:

Design the `boolean isEvenlySpaced(int x, int y, int z)` method, which receives three integers x, y, and z, and returns whether they are evenly spaced. Evenly spaced means that the difference between the smallest and medium number is the same as the difference between the medium and largest number.

Problem 4:

Design the `String cutTry(String s)` method, which receives a string *s* and, if *s* ends with `"try"`, it is removed. Otherwise, the original string is returned.

Problem 5:

In propositional logic, there are several *connectives* that act on boolean truth values. These include logical conjunction $\wedge$, disjunction $\vee$, conditional $\rightarrow$, biconditional $\leftrightarrow$, and negation $\neg$. We can represent *schemata* as a series of composed method calls. For example, an evaluation of

$$`P \rightarrow \neg(Q \leftrightarrow \neg R)'$$

where 'P' and 'R' are assigned to `false` and 'Q' is assigned to `true`, is equivalent to

```
static final boolean P = false;
static final boolean Q = true;
static final boolean R = false;

cond(P, not(bicond(Q, not(R))))
```

The presented schema resolves to `true`.

Design methods for the five connectives according to the following truth tables. These methods should be called `cond`, `bicond`, `and`, `or`, and `not`. Assume that T is `true` and F is `false`.

P	$\neg P$
T	*F*
F	T

Truth Table of '$\neg P$'

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Truth Table of '$P \wedge Q$'.

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Truth Table of '$P \vee Q$'.

P	Q	$P \to Q$
T	T	T
T	F	F
F	T	T
F	F	T

Truth Table of ‘$P \to Q$’.

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

Truth Table of ‘$P \leftrightarrow Q$’.

Problem 6:

Design the `boolean isInsideRectangle(double rx, double ry, double w, double h, double px, double py)` method that, when given a rectangle centered at (r_x, r_y), width w and height h as well as a point (p_x, p_y), returns whether the point is located strictly inside the rectangle.

Problem 7:

Carlo is shipping out orders of candy to local grocery stores. Boxes have a maximum weight defined by a value w, and we can (potentially) fit both small and large bars of candy in a box. Design the `int fitCandy(int s, int l, int w)` method that, when given a number of small bars s, large bars l, and maximum weight w, determines the number of small candy bars he can fit in the box. Large bars weigh five kilograms, and small bars weigh one kilogram. Note that Carlo always tries to fit large candies first before small. Return `-1` if it is impossible to fill the box with the given criteria. Below are some test examples. You cannot use loops, recursion, or data structures to solve this problem. Hint: consider this as an analysis of three cases.

```
fitCandy(4, 1, 9)      => 4
fitCandy(4, 1, 4)      => 4
fitCandy(1, 2, 6)      => 1
fitCandy(6, 1, 13)     => -1
fitCandy(60, 100, 550) => 50
fitCandy(7, 1, 12)     => 7
fitCandy(7, 1, 13)     => -1
```

Problem 8:

An IPv4 address contains four integer values stored in four octets, separated by dots. For instance, `192.168.1.244` is a valid IPv4 address. Another example is `149.165.192.52`. Design the `boolean isValidIpv4(String ip)` method that, when given a string, determines whether or not it represents a valid IPv4 address. Each octet must be an integer between zero and 255 inclusive. Note that some IPv4 addresses are, in reality, nonsensical, e.g., `0.0.0.0`, but we will not consider these as invalid. Below the examples is a helper method, `isNumeric`, to determine whether or not a string is "numeric." Understanding *how* this helper method works is unimportant for the time being. You *will* need to use `Integer.parseInt`, `substring`, and `indexOf`.

Warning: Again, you **cannot** use arrays, loops, or regular expressions to solve this problem, and if you do, you will receive a 0 on the entire problem set.

```
isValidIpv4("192.168.1.244")    => true
isValidIpv4("149.165.192.52")   => true
isValidIpv4("192.168.1.256")    => false
isValidIpv4("192.168.1201.23")  => false
isValidIpv4("192.168.1201.ABC") => false
isValidIpv4("ABC.DEF.GHI")      => false
isValidIpv4("192.168.1A6.201")  => false
```

```java
/**
 * Determines whether or not we can convert a given string into
 * an integer datatype.
 * @param n input string.
 * @return true if we can convert n to an int, false otherwise.
 */
static boolean isNumeric(String n) {
  try {
    Integer.parseInt(n);
    return true;
  } catch (NumberFormatException ex) {
    return false;
  }
}
```