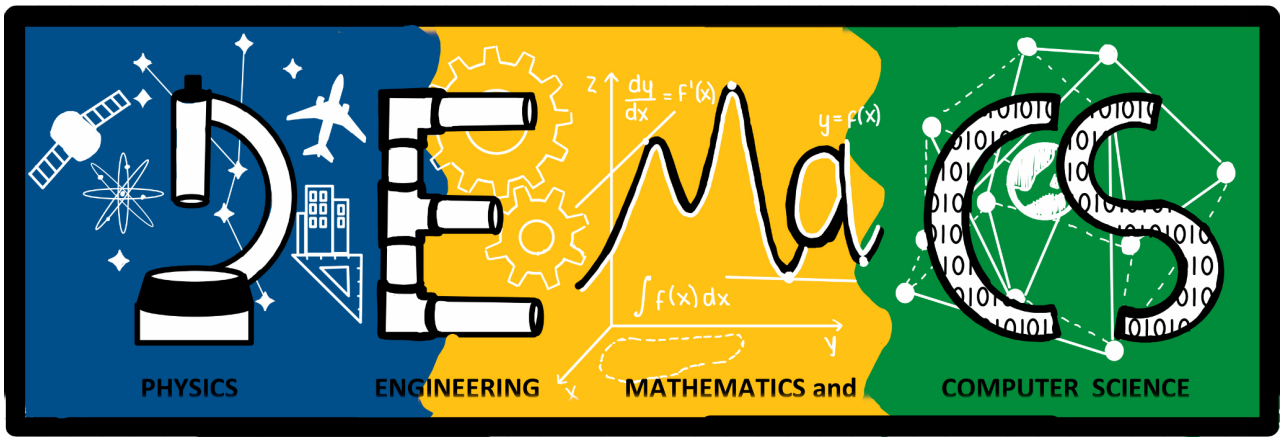

Elements of Computer Programming II (CSCI-121)

Skill Builder 2 - Decisions, Decisions, Decisions ...

Learning Outcomes

By the end of this activity, a student should be able to:

1. Use if-else statements for porper decision making in code.
2. Apply if-else statements in solution of real-world problems.
3. Implement methods that solve practical yet simple real-world problems.

Introduction

In this Skill Builder, you will be implementing methods that require logical or boolean expressions.

The Date Class

The Date class serves as a utility class with various methods that provide information and

calculations on dates.

Leap Year

Implement the `isLeapYear` method provided in the `Date` class. The method takes a single integer parameter that is the year and returns true if the year is a leap year; false otherwise.

In the Gregorian calendar, which is the calendar used by most modern countries, the following rules decide which years are leap years:

1. Every year divisible by 4 is a leap year.
2. But every year divisible by 100 is NOT a leap year
3. Unless the year is also divisible by 400, then it is still a leap year.

Name of Day of the Week

Implement the method `getNameOfDay`. This method takes an integer representing the day of the week and returns a string representing the name of the day of the week as given in Table 1 below.

Table 1: Numeric to String conversion of the days of the week.

Numerical Value	Name
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

If a numeric value is provided that is outside the 0 to 6 range, inclusively, then the number should be folded back into the range 0 to 6, inclusively. For example, if a value of 33 is provided, then it should be folded back to 5 because `33 % 7 = 5`, which in turn should result in `"Friday"`. Your method should handle negative integers as well. If a negative number is

provided, it should be wrapped back to the range 0 to 6, inclusively. To see how this can be done, consider the following image,

Notice, that -1 should be wrapped back to 6 (i.e. Saturday), equivalent to negative indexing in Python. Play with the modulo operator of negative integers in `jshell` and then figure out what calculation must take place to complete the mapping in the table below.

Table 2: Mapping negavite integer values to days of the week.

-1	6
-2	5
-3	4
-4	3
-5	2
-6	1

IMPORTANT: Use the string method `equalsIgnoreCase` as explained in class. For example, if you have a string variable called `month` and you want to see if it is equal to "thursday", ignoring case, then

```
"thursday".equalsIgnoreCase(month)
```

Converting the Name of a Month to an Integer Value

Implement the method `getMonthNumber`. This method takes a string representing the month's name and returns a number between 1 and 12, inclusively, representing the month's numeric value. If the string provided is not equal to any of the names of the months provided in the table below, then a -1 is returned.

The numeric values of the months is provided in the table below.

Table 3: Numeric values of the months of the year.

Numerical Value	Name
1	January
2	February
3	March
4	April
5	May
6	June
7	July
8	August
9	September
10	October
11	November
12	December

Convert the Month Integer Value to Name of Month

Implement the method `getMonthName`. This method takes an integer in the range 1 to 12, inclusively, and returns the name of the month. If the integer provided is not in the range 1 to 12, inclusively, then an empty string is returned.

The numeric values to the name of the month is provided in the Table 3 above.

IMPORTANT: Implement this method using a **switch** statement.

Days in a Month

Implement the overloaded method, `getNumberOfDaysInMonth`. The method with the signature `getNumberOfDaysInMonth(int,int)` takes a month value in the range 1 to 12, inclusively, and a year, and returns the number of days in that month in that year.

The method with the signature `getNumberOfDaysInMonth(String,int)` takes the name of a month and a year, and returns the number of days in that month in that year.

Both methods should return a -1 if the name of the month is not correct or the range is outside the range 1 to 12, inclusively.

For example,

```
Date.getNumberOfDaysInMonth(1, 2025)
```

results in `31`.

Also,

```
Date.getNumberOfDaysInMonth(1, 2024)
```

results in `29`.

Calculating the Day of Week

A `Date` class is really useless if it cannot provide the day of the week for a particular date. The next task is to provide this functionality. Before we implement the method, we must figure out an algorithm. To write down an algorithm, we need to figure out a quick method of calculating the day of the week for a particular date.

A year has 365 days unless it is a leap year, which would then be 366. Since 2023 is not a leap year, what day will January 1, 2024, fall on? The answer is 365 % 7 = 1. This is not the day of the week but rather the number of days after the day that January 1, 2023 falls on. January 1, 2023, falls on a Sunday (0). So, January 1, 2024, falls on 0 + 1 = 1 (Monday). This means that every January 1 of a nonleap year will start 1 day later than the previous year. If it is a leap year, it must start 2 days after the previous year started. So, January 1, 2025, will start on 1 (Monday) + 2 = 3 (Wednesday).

Since each year starts 1 day later if I know what day of the week January 1, year 1 starts on, then I should be able to calculate the day of the week that January 1, year n falls on. Let's assume this fact is known, then year n starts $n \% 7$ days after January 1 year 1. But that assumes that there are no leap years. Leap years add another day, so the formula should read,

$$\left(n + \left\lfloor \frac{n-1}{4} \right\rfloor\right) \% 7 \quad (1)$$

Where $\lfloor x \rfloor$ is the floor of x. The floor(x) returns the greatest integer less than x. Equation 1 adds

another day for each year, a leap year, but not every year divisible by 4, which is a leap year. Years divisible by 100 are leap years only if they are divisible by 400. Since every year divisible by 100 is also divisible by 4 (Can you show why?), the 2nd term equation in (eq. 1) overcounts the number of leap years. To correct that overcounting, subtract all the years that fall on the century mark (i.e., divisible by 100) and add back in the years divisible by 400. The final result is the equation,

$$\left(n + \lfloor \tfrac{n-1}{4} \rfloor - \lfloor \tfrac{n-1}{100} \rfloor + \lfloor \tfrac{n-1}{400} \rfloor\right) \% \ 7 \quad (2)$$

Now, let's use equation (2) to calculate the day of the week that January 1, 2024, falls on. For n=2024,

$$\left(2024 + \lfloor \tfrac{2024-1}{4} \rfloor - \lfloor \tfrac{2024-1}{100} \rfloor + \lfloor \tfrac{2024-1}{400} \rfloor\right) \% \ 7 = (2024 + 505 - 20 + 5) \% \ 7 = 1 \ ,$$

which is Monday. Work out what day January 1, 2026, will be.

Now, we have a reference point by which we can calculate the day of the week for any other date. To calculate the day of the week for January 17, 2024, one only needs to know what day January 1, 2024, falls on and then use the fact that January 17 comes 16 days later. So, January 17, 2024, occurs (16 % 7 = 2) days after the day that January 1, 2024, falls on, primarily 3 (Wednesday). What about March 12, 2024. January has 31 days, and February has 29 days (a leap year). So, March 12, 2024 occurs 31+29+12-1=71 days later. So, March 12, 2024 falls on a 1 + (71 % 7) = 2 (Tuesday).

A better way to calculate this without adding up the number of days that have passed since January 1 year n is to figure out how many days later each month starts after the previous month. We will call that the *offset*. January has 31 days, and 31 % 7 = 3. So February 1 occurs 3 days after where January 1 occurred. February has 28/29 days, depending on whether it is not or is a leap year. So for nonleap years, March 1 falls (28 % 7) = 0 falls on the same day of the week that February 1 fell on and 3 days after the day January 1 fell on. March has 31 days, so April 1 will fall 3 days later than when March 1 fell or 6 days relative to January 1. If we continue with this, we can generate a running offset, providing a quick lookup table for the 1st of any month. The table is as follows:

Table 4: Offset for each month.

Month	Offset	Offset for Leap Year
January	0	0
February	3	3
March	3	4
April	6	0
May	1	2
June	4	5
July	6	0
August	2	3
September	5	6
October	0	1
November	3	4
December	5	6

We now have a simple and fast method of calculating the day of the week for any date. As an example, consider April 13, 2026. We first calculate the day of the week for January 1, 2026.

$$\left(2026 + \lfloor\frac{2026-1}{4}\rfloor - \lfloor\frac{2026-1}{100}\rfloor + \lfloor\frac{2026-1}{400}\rfloor\right) \% \ 7 = (2026 + 506 - 20 + 5) \% \ 7 = 4$$

This is a Thursday. Now, according to Table 4, April 1st has an offset of 6 days. So, adding 6 to 4 [(6+4)%7=3 (Wednesday)]. April 13 occurs 12 days after the 1st, so we now have (3+12)%7=1, which is a Monday. Therefore, April 13, 2026, will fall on a Monday. Check it out!

You should try it out on September 22, 2030.

Month Offest

A private method is needed to provide the offsets in Table 4.

Implement a method called `getMonthOffSet` with two parameters, `month` and `year`. This method returns the offset per Table 4 if `month` is between 1 and 12, inclusively; otherwise -1.

Day of Week

You are now ready to implement the overloaded method `dayOfWeek`.

The first overloaded method with signature `public static int dayOfWeek(int, int, int)` has three paramters, `month`, which is an integer from 1 to 12, inclusively. A `dayOfMonth`, which is an integer within the range of days in `month`, and `year`.

The second overloaded method with signature `public static int dayOfWeek(String, int, int)` has three parameters with `month` being the name of the month and the other two paraemters as explained above.

IMPORTANT: The second overloaded method should use composition, primarily `getMonthNumber` and the first overloaded method of `dayOfWeek`.

Test Class

Once the `Date` class is implemented, test it using the included JUnit test. Configure the run configuration using Gradle. The IntelliJ unit tests does not include checks for specific implementation instructions, but CodeGrade will check for those specific implementations. **Please read the requirements for each method.**

Submitting Your Work

Once you have completed your work and have ensured that all requirements pass the unit test provided with this skill builder, submit your work on CodeGrade. Make sure that CodeGrade grades your work as expected.