3 - REPETITION

3.1 Loops in Java: The `for` loop, the `while` loop, and the `do while` loop

The third and last type of construct of programming languages allows for repetition of statements. This is called a loop, and there exist different types of loops that make programming easier depending on the situation. In Java, there are three different loops:

1. The `for` loop
2. The `while` loop
3. The `do while` loop

3.1.1 The `for` loop

This first type of loop is also known as a *counted* loop as it is used mostly when the programmer knows ahead of time how many times the statements need to be repeated. The structure of this loop is,

```
for (<initialization>; <condition>; <increment/decrement>) {
    <statements>
}
```

For example,

```
for (int i = 0; i < 10; i++) {
    Stdout.println(i);
}
Stdout.println("End of job.");
```

This example of a loop starts by declaring a variable `i` and setting it to an initial value of zero. The condition of the loop is tested. If the condition is true, then the statements within the braces are executed. At the end of the statements, the variable `i` is incremented by one by means of the expression `i++`. Then the condition is tested again. If it is true then the statements are executed again. Then the variable `i` is incremented once again. The process continues until the condition becomes false at which point execution proceeds with the statement immediately after the loop. In the case above the output would be,

```
0
1
2
3
4
5
6
7
8
9
End of job.
```

As mentioned above, the variable `i` is incremented by means of the expression `i++`. This is equivalent to `i = i + 1`. We can write other expressions that make the variable increment by different values. For example,

```
for (int i = 1; i < 10; i = i + 2) {
    Stdout.println(i);
}
Stdout.println(“End of odd numbers.”);
```

would result in,

```
1
3
5
7
9
End of odd numbers.
```

Another example,

```
for (int i = 5; i >= 1; i--) {
    Stdout.println(i);
}
Stdout.println(“End of backward numbers.”);
```

would result in,

```
5
4
3
```

```
2
1
End of backward numbers.
```

3.1.1 Exercises

1. Write three separate loops in a Java program:

   a) To print numbers from 1 to 20.
   b) To print odd numbers from 1 to 50.
   c) To prints odd numbers backwards from 20 to -10

2. Write a Java program that prints out a table of numbers from 1 to 5 with their squares and cubes. Format the table using appropriate field widths. The output would look like this:

```
Number    Square       Cube
     1         1          1
     2         4          8
     3         9         27
     4        16         64
     5        25        125
```

3. Write a Java program that asks the user for 10 numbers. The program prints the largest number. The entire program, including import statements and declarations, can be written with only 14 lines of code.

4. Using a counted loop, write a Java program that prints the following:

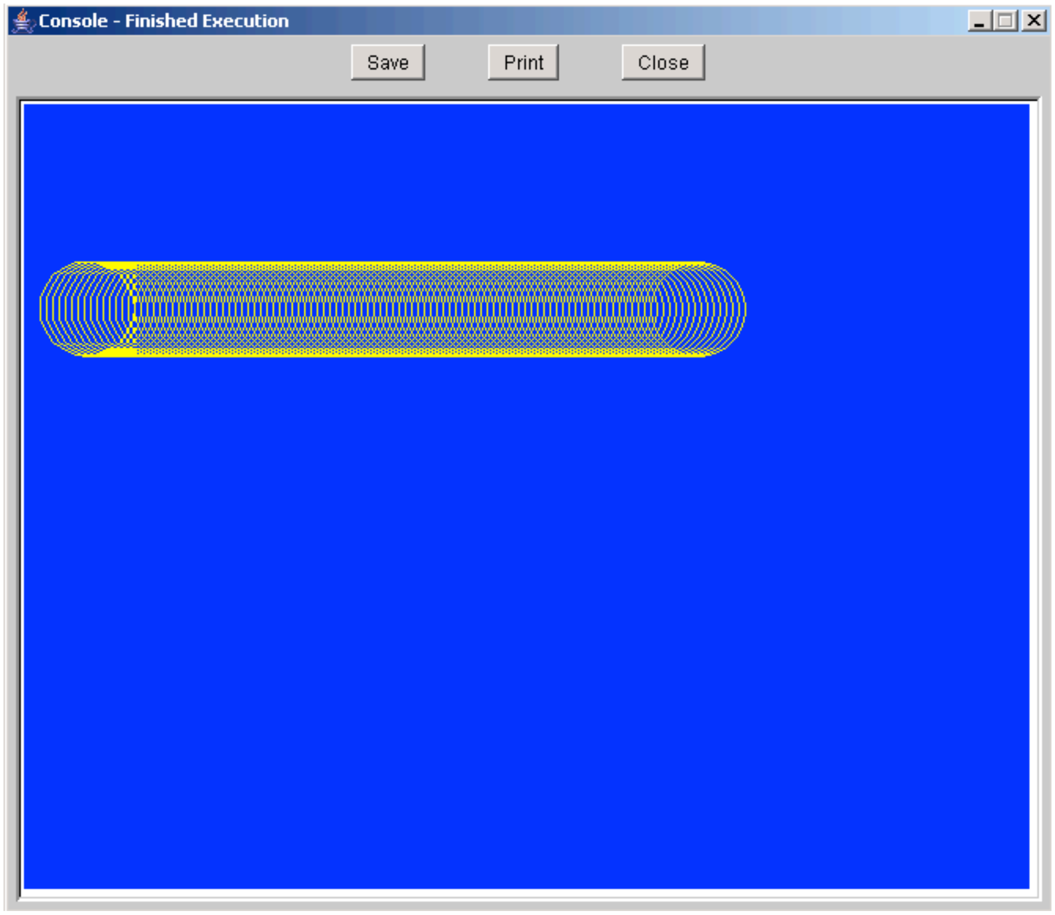
```
1 + 1 = 2
2 + 2 = 4
3 + 3 = 6
4 + 4 = 8
```

5. Write a Java program that asks the user for a number and then prints its first 8 multiples. For example:

```
Enter a number: 6
The first 8 multiples of 6 are: 6, 12, 18, 24, 30, 36, 42, 48
```

6. Write a Java program that draws 100 overlapping yellow circles on a blue background, side by side. The distance between one circle and the next

needs to be 6 pixels, and the diameter of the circles is 60 pixels. The output looks like this,



7. Write a Java program that prints all numbers from 1 to 20 except the multiples of 3

8. Write a Java program that prints out the calendar for August 2015, skipping Saturdays and Sundays. The output looks like this,

```
August 2015
M  T  W  Th F  M  T  W  Th F  M  T  W  Th F  M  T  W  Th F  M
3  4  5  6  7  10 11 12 13 14 17 18 19 20 21 24 25 26 27 28 31
```

9. Modify your program of question 8 to include Saturdays and Sundays in standard format. Make sure to right justify numbers. The output is,

```
August 2015
```

```
Sun Mon Tue Wed Thu Fri Sat
                          1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

10. Write a Java program that prints out the first fifty Fibonacci numbers. The pattern for these numbers is,

```
1    1    2    3    5    8    13   21   34   ...
```

The general formula for the *Fibonacci sequence* is

$$t_1 = 1$$
$$t_2 = 1$$
$$t_n = t_{n-1} + t_{n-2}, n > 2$$

For more advanced programmers, be sure to write this program with iteration, not recursion.

11. Write a Java program that prints all the factors of a number given by the user. Make this program as efficient as possible by minimizing the upper limit of the loop. For example, if the given number is 400, the loop does not need to test all numbers from 1 to 400, but from 1 to a much smaller number.

12. Write a Java program that asks the user for a number and determines if the number is prime.

13. Write a Java program that, from a given month and a given year, calculates the weekday for the first day of that month. For example, given the month 11 and the year 2015, the program calculates that the first day of the month is a Sunday. The programs gives this result in the form of a number where 0 is Sunday, 1 is Monday, 2 is Tuesday, etc.

This question is best done in two parts:

Part 1
Calculate the weekday for the 1st of January of the given year. To do this, you can use Gauss's formula[1]:

$$R(1+5R(A-1,4)+4R(A-1,100)+6R(A-1,400),7)$$

where *R* is the modulus (remainder) operation and *A* is the given year. This formula will return 0 for Sunday, 1 for Monday, 2 for Tuesday, etc.

You can use the following table to test this part of the exercise:

Year	January 1st
2015	4
2000	6
2076	3
1993	5
1827	1
1943	5

Part 2
Once you have the weekday for January 1st of the given year, you can use a loop to count up to the first day of the given month. You will need to keep track of the months that have 31, 30, 29, or 28 days.

You can use the following table to test this part of the exercise:

[1] https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week#Gauss.27_algorithm

Year	Month	1st day of month
2015	11	0
2004	2	0
1991	6	6
2025	8	5
1801	4	3
1957	7	1

3.1.2 The `while` loop

This loop is used when the programmer does not know in advance how many times the statements inside the loop will be executed. Rather, the statements are executed until a given condition becomes `false`. The structure of this loop is

```
while (<condition is true>) {
    <statements>
}
```

The following program fragment keeps asking the user for a test mark out of 40 and prints it as a percentage until the user types in a mark of `-1`. This is the indicator that the loop comes to an end:

```
double mark = -1;
Stdout.print("Enter mark out of 40 (-1 to quit): ");
mark = Stdin.readDouble();
while (mark != -1) {
    Stdout.println("Mark is " + (mark / 40) * 100 + "%");
    Stdout.print("Enter mark out of 40 (-1 to quit): ");
    mark = Stdin.readInt();
}
Stdout.println("End of marks.");
```

A sample run is,

```
Enter mark out of 40: 30
Mark is 75.0%
Enter mark out of 40: 28
Mark is 70.0%
Enter mark out of 40: -1
End of marks.
```

Notice the placement of the assignment statements prior to testing the condition of the loop. It is crucial that the variables that are being tested have values in them. Otherwise the program will not compile, but in some languages such as C, the program will compile and run but with unpredictable results.

Also notice that these assignment statements occur again inside the loop. This is necessary to ensure that the variable being tested is changing in value, thereby guaranteeing that the loop will eventually come to an end. It is possible that the starting condition is `false` right from the beginning, which means that this loop might never be executed. In contrast, the next type of loop guarantees at least one pass through the loop.

3.1.3 The `do while` loop

As just mentioned, the `do while` loop is similar to the `while` loop with the difference that the the `do while` loop guarantees at least one pass through all the statements contained inside of it.

Also, the condition of the loop is placed at the end of the loop instead of the beginning which, at times, makes more semantic sense, i.e., it is more the way that we as programmer would think. The structure of the `do while` loop is,

```
do {
   <statements>
} while (<condition is true>);
```

An example is,

```
int number = -1;
do {
    Stdout.print(“Enter a number (-1 to quit): “);
    number = Stdin.readInt();

    if (number % 3 == 0)
        Stdout.println(“3 is a factor of “ + number);
    else
        Stdout.println(“3 is not a factor of “ + number);
} while (number != -1);
Stdout.println(“End of program.”);
```

A sample run is,

```
Enter a number (-1 to quit): 30
3 is a factor of 30
Enter a number (-1 to quit): 10
3 is not a factor of 10
Enter a number (-1 to quit): -1
3 is not a factor of -1
```

```
End of program.
```

In this loop, note that the value of `-1` also gets tested before exiting the loop, whereas in the `while` loop this is not the case.

All three types of loops that we have learned have different structures that make it more convenient for programming and also allow for better readability and understanding of programs.

3.1.2 and 3.1.3 Exercises

1. Write a Java program that asks the user for a number and prints whether the number is even or odd. The program keeps asking for a number until the user types `-1`, at which point the program ends.

2. Write a Java program that keeps asking the user for numbers. The program stops when the user enters `-99`. Then the program prints out the sum of all the entered numbers not including the `-99` in the sum.

3. Write a Java program that keeps asking for a word until the user types `quit`, at which point the program prints the word with the lowest alphabetical value, not including the word `quit`. The program then ends. You can use the method `equalsIgnoreCase()` so that Java will not care about upper or lower case. All these will be the same: `QUIT`, `quit`, `Quit`, or any combination of capitals and lower case letters.

4. Write a Java program that asks the user for a command. The program prints the words `valid command` only if the user enters a command that is exactly one letter long and the letter is a capital letter. Otherwise the program prints `invalid command` and asks the user for a letter again. The program quits when the user enters `Q`. We can determine the length of string with the method `length()`. For example, if the string variable `word` contains the text `Hello`, then the `word.length()` returns 5. To determine if the letter is upper case, we can use the ASCII table.

5. Computer game: Create a new console window. Then, have the computer ask you to enter a number. After this, clear the screen. The program stores the number and another person will attempt to guess the number that has been stored. The program then asks the player to make a guess. If the guess is larger than the stored number, the program will

print on the screen `Try a smaller number`. If the number guessed is smaller than the original number, the program will print on the screen `Try a larger number`. The program will continue doing this until the user guesses the stored number correctly. Have your program keep track of the number of times that the player makes a guess, and output this at the end. A sample run is,

```
Enter number to guess: 36
. (The screen is cleared here so that the player
. (does not see the number entered)
Enter a guess: 30
Your guess is too small. Try a larger number
Enter a guess: 39
Your guess is too high. Try a smaller number
Enter a guess: 35
Your guess is too low. Try a larger number
Enter a guess: 36
You guessed in 4 tries
```

6. Modify the previous program so that the computer generates the number to be guessed internally by using a pseudo-random number generator. Use the method

```
Math.random()
```

which generates a random number in the range [0, 1). We can generate a random integer in the range [0, upperLimit) with this operation:

```
(int) (Math.random() * <upperLimit>)
```

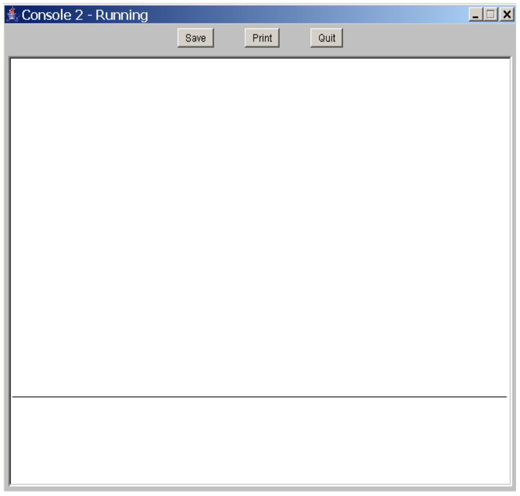
For example, the statement

```
int numberToGuess = (int) (Math.random() * 40);
```

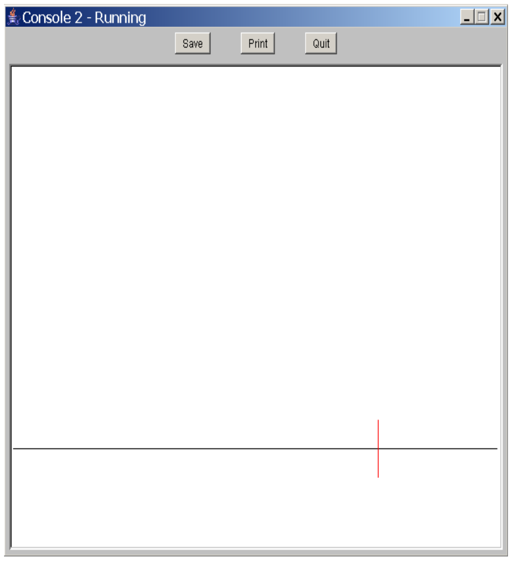
will assign to the variable `numberToGuess` a random integer in the range `[0, 40)`.

7. Add a graphical component to the program so that a visual effect gives the player a sense of where the numbers are. The program will draw a horizontal line at the start of the program on a separate console and then draw colour vertical lines to show the player where they are in the guessing game. Here are the steps:

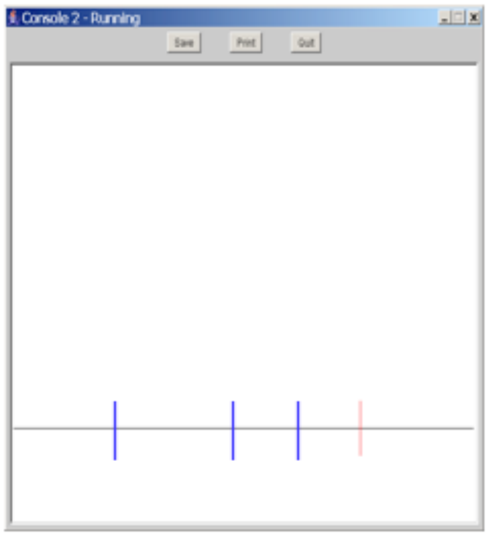
Draw a horizontal black line close to the bottom of the screen:



Generate a random number in the range 0 to `maxx()`. This is the number that we will attempt to guess. Draw a red vertical line on the spot on the line where the number is represented:



Ask the user to make a guess: A number between 0 and `maxx()`. After each guess, draw a blue vertical line on the spot on the line where the number is represented. This keeps happening until the number is guessed:



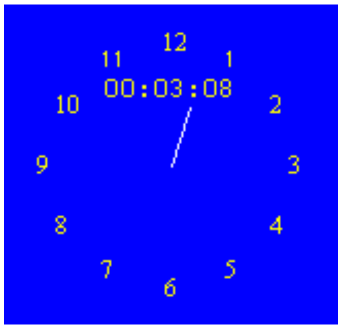
8. Extending: Write a Java program that tells the player what is the next best number to guess. If the number to be guessed is between 0 and 1000, it can be guessed in no more than 11 guesses. Write the program so that in 11 guesses or less the number is found.

9. Write a Java program that displays a digital stopwatch with minutes, seconds, and tenths of seconds. The method

   ```
   c.setCursor(row, column)
   ```

   moves the cursor to the desired location on the screen. A print statement after moving the cursor to that location will cause the output to print at the location. This way we can always print on the same location without clearing the screen. To simulate the delay of the watch, write a delay loop. We will see how to delay with the correct timing soon.

10. Implement the stopwatch of the previous question as an analog stopwatch that shows the seconds hand along with the seconds and tenths of seconds in digital format. For example:



More Exercises on Loops

1. Write a Java program that prints a table of values for the function $f(x) = x^2$ where $-3 \le x \le 3$. The program output is:

```
x        f(x)
-----------
-3       9
-2       4
-1       1
 0       0
 1       1
 2       4
 3       9
```

2. Write a Java program that prints a table of values for the function $f(x) = 3x^2 - 1$ where $-2 \le x \le 2$. The program output is:

```
x        f(x)
-----------
-2        11
-1         2
 0        -1
 1         2
 2        11
```

3. Write a Java program that prints a table of values for the function $f(x) = 2x^3 + x^2 - x + 6$ where $-5 \le x \le 5$. The program output is:

```
x        f(x)
------------
-5       -214
```

```
-4      -102
-3       -36
-2        -4
-1         6
 0         6
 1         8
 2        24
 3        66
 4       146
 5       276
```

4. Write a Java program that prints all perfect square numbers from 1 to 225. The program output is:

```
1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225
```

5. Write a Java program that draws a dot in a random location on the screen. To calculate a random x-coordinate and a random y-coordinate use the statements

```
int x = (int) (Math.random() * c.maxx());
int y = (int) (Math.random() * c.maxy());
```

   Then use the `x` and the `y` to draw a line that starts and ends at the same location. This will draw a point:

```
c.drawLine(x, y, x, y);
```

6. Write a Java program that uses the code of the previous exercise. This new program draws one thousand dots at random locations.

7. Modify the previous program so that the dots are of a random colour. To get a random colour use the following:

```
int colour = (int) (Math.random() * 16);
```

8. Consider the following program,

```
import hsa.*;
public class Multiples {
    public static void main(String[] args) {
        Stdout.print("Enter a number: ");
        int number = Stdin.readInt();
```

```
            for (int multiple = 1; multiple <= 10; multiple++) {
                Stdout.print(number * multiple + " ");
            }
        }
    }
```

a. Rewrite the program using a `while()` loop
b. Rewrite the program using a `do while()` loop

9. The code inside a loop needs to be indented one level. The following program contains commands we will see soon, but we should be able to indent it nevertheless. Indent the following program :

```
import hsa.*;

public class Dots {
public static void main (String[] args) {
int numberOfDots = 0;
String ipAddress = "192.168.2.1";

if (ipAddress.length () != 0) {
if (ipAddress.charAt (0) != '.') {
if (ipAddress.charAt (ipAddress.length () - 1) != '.') {
for (int i = 0 ; i < ipAddress.length () ; i++) {
if (ipAddress.charAt (i) == '.') {
numberOfDots++;
}
}
}
}
}
if (numberOfDots == 3) {
Stdout.println(true);
}
else {
Stdout.println(false);
}
}
}
```

10. Extending Exercise.

Graphing *y = sin(x)* on the interval [-360, 360]

Task. In this exercise we will write a Java program that graphs the function

$$y = \sin(x), -360 \le x \le 360$$

We will print the table of values in one window, and graph the function on another window.

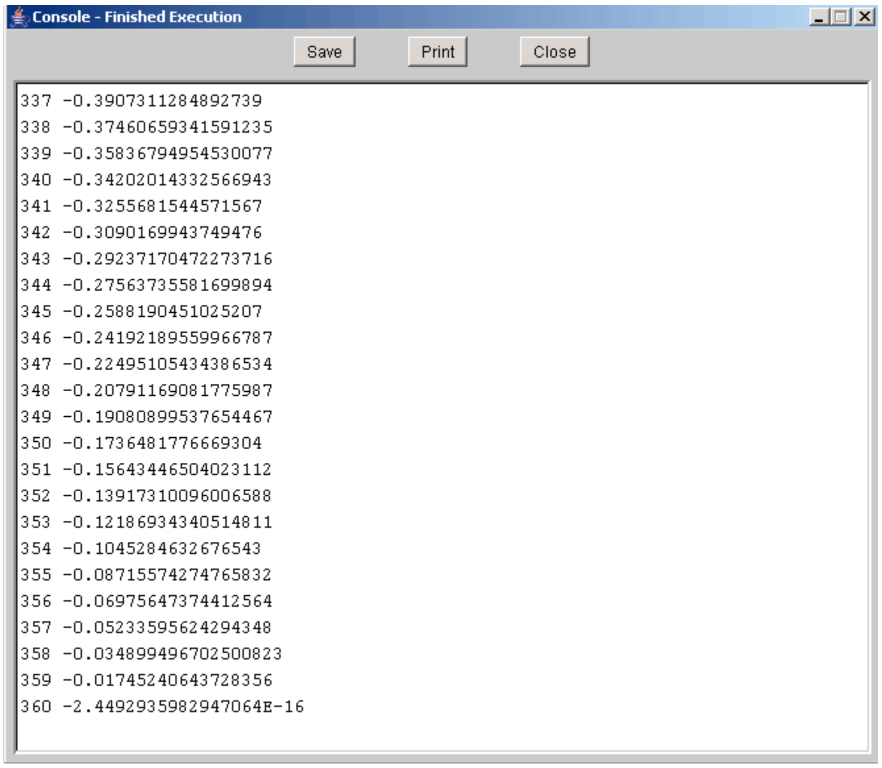
Process. Implement the following algorithm using Java:

```
declare Console c
declare Console d

draw the x-y  axes in console d, with the origin in the centre of
the window

change the color to blue in console d

for (x running from -360 to 360, incrementing by 1) {
     print in Console c the value of x and the value of y
     plot the point in Console d
}
```

Important notes to consider.

1. To calculate the value of `y` we use the method `Math.sin(x)`. However, `x` is in degrees and the method `Math.sin()` expects radians. We need to use the method `Math.toRadians(x)` within `Math.sin()` in a similar way as when we used `Math.toDegrees(x)` in our analog clock program.

2. When printing the values, the `x` and the `y` print side by side in a simple table as shown in the sample below:



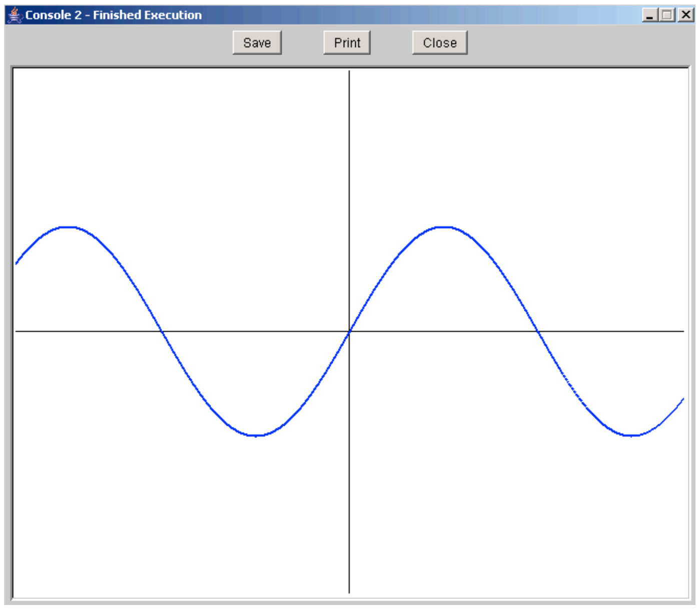
```
337 -0.3907311284892739
338 -0.37460659341591235
339 -0.35836794954530077
340 -0.34202014332566943
341 -0.3255681544571567
342 -0.3090169943749476
343 -0.29237170472273716
344 -0.27563735581699894
345 -0.2588190451025207
346 -0.24192189559966787
347 -0.22495105434386534
348 -0.20791169081775987
349 -0.19080899537654467
350 -0.1736481776669304
351 -0.15643446504023112
352 -0.13917310096006588
353 -0.12186934340514811
354 -0.1045284632676543
355 -0.08715574274765832
356 -0.06975647374412564
357 -0.05233595624294348
358 -0.034899496702500823
359 -0.01745240643728356
360 -2.4492935982947064E-16
```

3. To plot each point, use the method `drawOval()` with a diameter of 1. For example: `d.drawOval(x, y, 1, 1)`

4. Be sure to apply the correct transformations to the `x` and `y` coordinates so that the point is plotted in the correct location.

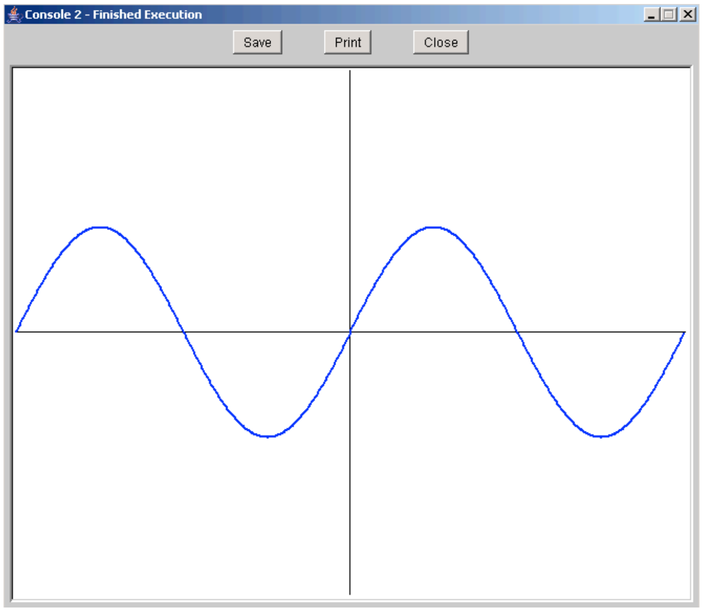
   a) The `x` coordinate needs to be shifted to the right by half the width of the window.
   b) The `y`-coordinate needs to be stretched vertically by a factor of 100 in order for it to be visible, then reflected on the `x`-axis, then shifted down by half the height of the window.

When done, the output will look like this:

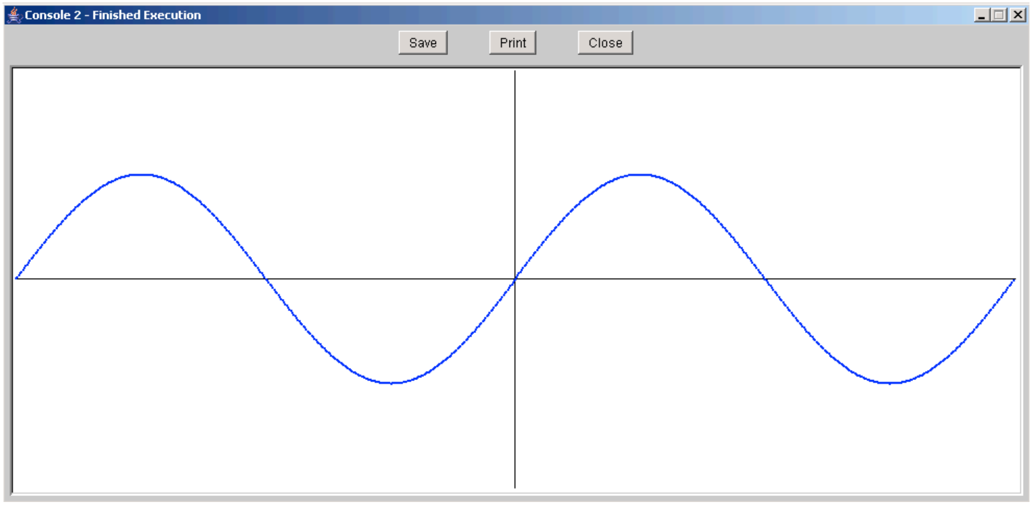


Further considerations

Since the window is 640 pixels wide, the graph shown above is approximately in the interval [-320, 320]. The graph needs to be stretched/ compressed horizontally so that the full two cycles of the graph are shown.



Make a transformation so that if the window changes width, the graph readjusts automatically to always show the two cycles. Note that this needs to be a horizontal transformation of the graph. The loop still needs to run from -360 to 360.



11. Copy and indent the following program.

```
import hsa.*;
public class Exercise11 {
public static void main (String[] args) {
int minutes = 0;
int seconds = 0;
while (true) {
if (minutes < 10) {
Stdout.print (0);
}
Stdout.print(minutes);
Stdout.print(":");
if (seconds < 10) {
Stdout.print (0);
}
Stdout.println(seconds);
seconds = (seconds + 1) % 60;
if (seconds == 0) {
minutes = (minutes + 1) % 60;
}
}
}
}
```