Part 1 (0 points): Warm-up

Do **not** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs or instructors during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Write a method `printTypeOfChar`. This method should take as input a `char` and *print*. If the `char` is upper case, your method should print `UPPERCASE`. If the `char` is lower case, your method should print `LOWERCASE`. If the char is any other symbol, your method should print `SYMBOL`. Note that it is *not* necessary to make 52 different cases if you consider the order of the Unicode chart and the fact that you can use the $<$ and $>$ operators on them.

Warm-up Question 2 (0 points)

Write a method `countUppercase`. Your method should take as input a `String` and return an `int` representing the number of upper case letters in the `String`. Now add a `main` method where you use the `Scanner` class to read a `String` from the user and call your method, outputting the returned result.

Warm-up Question 3 (0 points)

A prime number is a positive number whose only even divisors are 1 and itself. Write a method `isPrime` that takes as input an integer `n` and returns a `boolean` representing whether `n` is prime or not. Make sure to handle cases where the integer is negative. (Your method should return false in these cases.)

Warm-up Question 4 (0 points)

Write a method `firstPrimeNumbers` which takes as input an `int n` and returns an `int[]`. The `int[]` should contain the first `n` prime numbers.

Warm-up Question 5 (0 points)

x is a factor of y if y is a multiple of x. Write a method `calculateFactors`. The method should take as input an `int n` and return an `int[]` containing all the factors of the number `n`.

Part 2

The questions in this part of the assignment will be graded. Note the test program `AssignmentTwoTests.java` *provided with the assignment. There are instructions on how to run the test code after the assignment specification below. You must run your code through these tests. It is OK if your code does not pass every test (although you will lose some marks for this), but your code* ***must*** *compile with the test file. The TAs may use a test program with additional tests to the one we provided, so you should run additional tests. The point of the test file is to help make sure you followed the specifications correctly.*

You will not be required to hand in a `main` *method, but you should write one in order to test your code properly. You are also encouraged to add additional auxiliary methods to the required ones below. This will help reduce errors, allow for you to test your code more easily, and make your code more readable and understandable.*

Question 1: Managing an online store (60 points)

In this question, you will design some methods that are part of an online store. The following code should go into a class called `OnlineStore` .

You should write the following methods:

- *When you purchase a product on* `Amazon.com` *with a credit card,* `Amazon.com` *performs this simple credit card verification first. Then, it does a more thorough check before sending you the product. In this question, you will write the code for the simple, initial verification.*

  Write a method called `validateCreditCard` which takes as input an `int[]` representing the 16 digits of a credit card and returns a boolean. It should calculate whether the credit card has a valid number (see below) and return `true` if it is valid and `false` if it is not valid.

  The `int[]` contains the digits of the credit card. The first spot in the array contains the first digit, the second spot contains the second digit, etc.

  For example, the credit card number 1234 5678 1234 5678 would be stored in an array:

digit	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
arrayindex	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

  A credit card is valid if the `int[]` satisfies the following conditions:

  1. The length is `16` .
  2. Every digit in the array is between 0 and 9 (inclusive)
  3. The *checksum* computation results in a multiple of 10 (see below).

  If all three conditions hold, then your method should return `true`. If one of the conditions fails, your method should return false. (*Hint: These 3 different conditions are excellent candidates for auxiliary methods, one for each of them.)*

  To compute the checksum, you should perform the following steps:

  1. For every digit that appears at an *even* index, double the value.
  2. Now for every digit in the array, calculate the sum of it's digits. (Hint: For numbers less than 10, this will just be the number itself. For numbers bigger than 10–i.e. those that had been doubled– you can use integer division and the modulo operator to figure out the individual digits.)
  3. To get the result of the checksum computation, add up the numbers that remain.
  4. If the sum is a multiple of 10, then the checksum is valid. Otherwise, it is not valid and the card is not valid.

As an example of computing the checksum, consider the above example of validating the card with numbers `1234 5678 1234 5678`

1. First double all the values occurring at even indices:

$originaldigit$	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
$afterdoubling$	2	2	6	4	10	6	14	8	2	2	6	4	10	6	14	8
$arrayindex$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2. Next, for each number in the array, calculate the sum of it's digits:

$originaldigit$	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
$afterdoubling$	2	2	6	4	10	6	14	8	2	2	6	4	10	6	14	8
$sumofdigits$	2	2	6	4	1	6	5	8	2	2	6	4	1	6	5	8
$arrayindex$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

3. Now add up all the values that are left: In this case, it sums to 68. Since 68 is not a multiple of 10, this credit card number is not valid.

If you have a Visa or Mastercard, you should be able to test your card's validity this way.

- `validateIsbn` : Write a method that checks whether a `String` is a valid ISBN-10 number or not. Your method should take as input a `String` and return a `boolean` value of `true` if the `String` is a valid ISBN and `false` otherwise. Note that we will assume for simplicity that there are no dashes or other characters in the `String`. The `String` is only the digits of the ISBN number itself and does not include any dashes or spaces.

  In order for a `String` to be a valid ISBN, it must satisfy the following:

  1. It should have a length of exactly 10
  2. The first 9 characters should be digits ranging from 0-9
  3. The last character should be either a digit ranging from 0-9 OR an X. Note that a lower case x is not valid.
  4. The ISBN must also satisfy a checksum, which is described below.

  If all of the above are `true`, then your method should return `true`. If one or more of them are false, then your method should return `false`.

  To compute the checksum this time, you should perform the following:

  1. For each of the first 9 digits in the ISBN, multiply it by $10 - theStringindex$. For example, you should multiply the first digit (the 0th index of the `String` by 10, the second digit by 9, etc.
  2. Add these results together.
  3. If the last digit is 0-9, add it to the previous step. If the last digit is an X, then add 10 to the previous step.
  4. If the result is a multiple of 11, then the `String` is a valid ISBN. If the result is not a multiple of 11, then the `String` is not a valid ISBN.

Question 2: Detecting whether a point is inside a shape (40 points)

In this question you will use geometric reasoning and apply a given ***algorithm*** *to calculate whether a point is inside of various shapes or not. All of your code for this question should go into a file* `GeometryTools.java`

In the process of developing simulations and games, a very important issue is being able to detect whether a point is inside of a polygon or not. This aids in things such as collision detection. For example, if you

were programming the game of Mario, you would want to check whether he and an enemy overlapped. In this question, you will write two methods, one to detect whether a point is inside of a circle and another to detect if a point is inside a convex polygon.

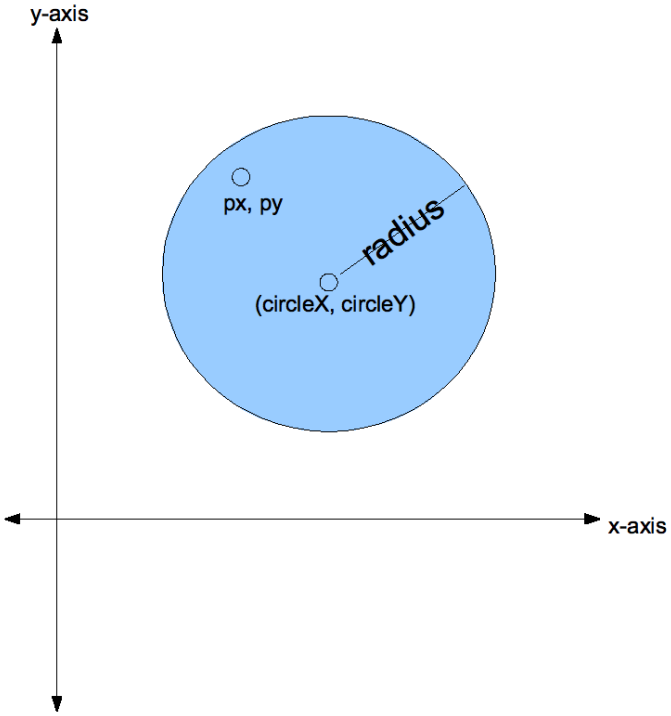


Figure 1: Your method detectInCircle should return true whenever the distance from the point (circleX, circleY) to (px,py) is less than the radius of the circle

- Write a method
  `detectInCircle(double px, double py, double circleX, double circleY, double radius)`
  .
  `px` and `py` represent the x and y coordinates, respectively, of the point you are considering. `circleX`, and `circleY` represent the x and y coordinates, respectively, of the circle's center. `radius` represents the distance from the center of the circle to the circumference of the circle.

  Your method should return a `boolean` value of `true` if the point is inside of the circle and false otherwise. Due to the imprecision of floating point arithmetic, we will not worry about the case where the point is exactly on the perimeter, but we will use a constant `EPSILON` to make your method less susceptible to rounding errors (see below).

  To detect whether a point is inside of a circle or not, you can check whether the distance from the point to the center of the circle is less than the radius of the circle or not.

  To calculate the distance, one would use the following formula:

  $$distance(px, py, circlex, circley) = \sqrt{(px - circlex)^2 + (py - circley)^2}$$

  If this is less than *radius* then the point is inside the circle and the method should return `true`. Otherwise it should return `false`.

Calculating the square-root on a computer is actually a bit slow and unnecessary. Since we only care whether it is greater than or smaller than `radius`, and don't care by how much, it is actually better to check whether the distance-squared is less than `radius` squared or not.

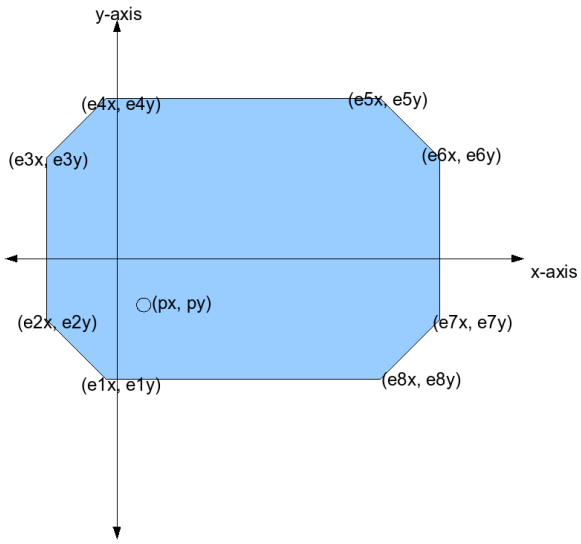
In otherwords, your method only needs to check whether

$$(px - circlex)^2 + (py - circley)^2 < radius^2$$

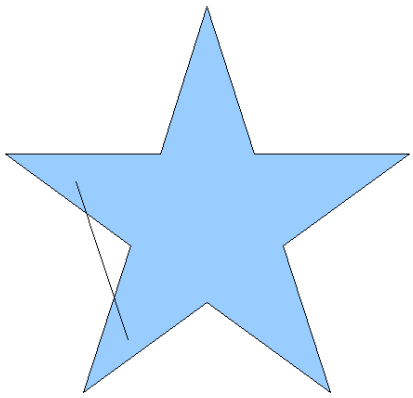
In order to avoid rounding issues when the point is near the circumference, you should define a *class* constant at the top of the class `GeometryTools` called `EPSILON` and give it value .0001 . Since `EPSILON` is a class constant, it should not be a part of any method but should be a part of the class `GeometryTools`. The test program (see below) will verify this is done correctly.

Your actual comparison should then be

$$(px - circlex)^2 + (py - circley)^2 < radius^2 + EPSILON$$



(a) An example of a convexpolygon. Your method detectInPolygon should return true whenever the point is inside the polygon and false otherwise



(b) An example of a concave polygon. You can draw a line segment that starts inside the polygon, goes outside, and enters the polygon again. Your code does not need to worry about this case.

- Write a method `detectInPolygon(double px, double py, double[] xCoords, double[] yCoords)` This method should detect whether a point is inside of a convex 2d polygon. Your method should take as input 4 things. `px` and `py` once again refer to the point in question.

  The third and fourth parameters are each arrays of doubles. The two arrays are conceptually linked in that the same index can be used in the `xCoords` and `yCoords` to get the coordinates of one point. (The computer does not know this of course. Side note: When we learn about creating our own types, we will see a better way to do this.)

These coordinates represent the corners of a polygon in either clockwise or counterclockwise order. In the diagram above and to the left, one array would have the values $\{e1x, e2x, e3x, e4x, e5x, e6x, e7x, e8x\}$ and the other would have the values $\{e1y, e2y, e3y, e4y, e5y, e6y, e7y, e8y\}$. The method should return `true` if the point is inside the polygon and `false` otherwise. It does not need to worry about cases where the point is on the edge of the polygon.

There are several ways to detect whether a point is inside of a polygon or not. In the case of a *convex* polygon, you can do this using the following technique:

Given a list of all the corners of the polygon *in order, either clockwise or counter-clockwise* and the point you wish to detect (px, py), you should do the following (see diagram):

1. Calculate the *vector* v from the 1st point $(e1x, e1y)$ to the 2nd point $(e2x, e2y)$. To do this, you should compute $v = (e2x - e1x, e2y - e1y)$. You should store this coordinate using two variables of type `double`.
2. Calculate the *vector* from $(e1x, e1y)$ to (px, py) using a similar subtraction. $u = (px - e1x, py - e1y)$

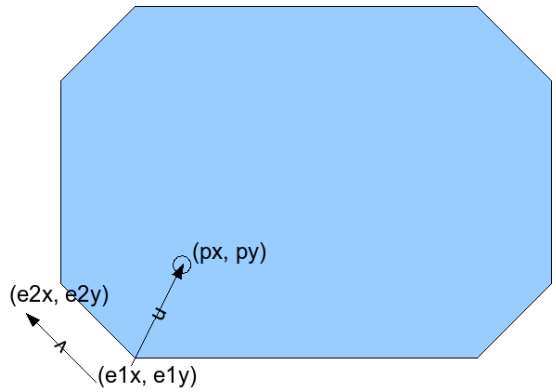


Figure 2: You should calculate the vector v by computing $(e2x, e2y) - (e1x, e1y)$. You should calculate the vector u by computing $(px, py) - (e1x - e1y)$

3. Now calculate the sign of the cross product (see below) of u and v. Note whether this sign is positive or negative.
4. Next, work your way around the polygon by repeating this computation using the 2nd and 3rd points in the arrays instead of the 1st and 2nd. **Make sure that you are consistent in your ordering and compute the 3rd point minus the 2nd point. Otherwise your results will be wrong.** Again, note whether the result is positive or negative.
5. Repeat this procedure for every point in the array. When you reach the final point, you should pair the last point with the first point. In this case too, be careful to make sure you calculate the first point $(e1x, e1y)$ minus the last point (enx, eny).
6. If all of the cross product signs are the same (either all are positive or all are negative), then the point is *inside* the polygon. Otherwise, it is outside the polygon.

To compute the *sign of the cross product*, you can use the following formula. Denote by v_x and v_y the x and y coordinates of v respectively and denote by u_x and u_y the x and y coordinates of u, respectively.

$$sign(v,u) = \begin{cases} +1 & v_x * u_y - v_y * u_x > 0 \\ -1 & v_x * u_y - v_y * u_x < 0 \\ undefined & v_x * u_y - v_y * u_x = 0 \end{cases}$$

Note that for this question you do not need to worry about `EPSILON` as in the previous section. Also note that since you can assume the point is not on the edge of the polygon, that the sign computation will never equal 0 exactly.

You may assume that the arrays of points are given in either a clockwise or a counter clockwise order. You may also assume that both arrays have the same length and that each array has a size of at least 3. In other words, it does not matter what your code does if these assumptions do not hold.

Verifying your code

Run your code against the test programs to verify it. To do this, you should make sure that `OnlineStore.java`, `GeometryTools.java` and `AssignmentTwoTests.java` are all in the same folder.

Then, compile all of them together by typing

```
javac *.java
```

If the program does not compile, it means you are either missing a public method or constant or one of the required public methods does not take the correct arguments. Your code must compile with these test cases. If you are not able to do so, you should ask a TA or instructor for help. After this, you may run the test program by typing

```
java AssignmentTwoTests
```

What To Submit

`OnlineStore.java`
`GeometryTools.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.