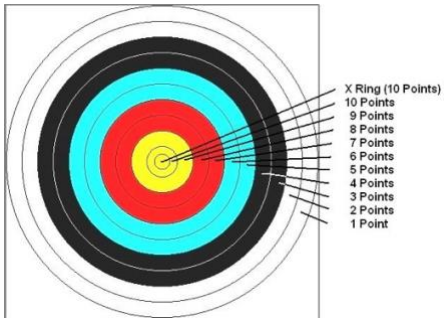
Archery competition

You are asked by a little start-up to write a program that can be used during an archery competition. Another software engineer has been asked to create a mobile application that is used by the judges for entering the scores for each archer, so you can focus on the back-end of the application.
If you are able to write the program well it might be a goldmine for the start-up and you, as the creator of this program, can of course share in the profits. Because there is little time left for developing the application, which is common practice for start-ups, you much write (almost) bug free. Unfortunately, the mobile app is still in development, so you have to use have some software that acts as the mobile app and that will generate the results for each archer. This code is almost ready and needs some last-minute adjustments and improvements.

Archers shoot during a competition of 10 rounds. Per round they shoot 3 arrows. So in total each archer will shoot 30 arrows. Depending on the exact kind of competition the ranking can be determined using different schemes. This means that it is important to keep track of each arrow (points) in each round. (You should not let an archer shoot 30 arrows and then only register the total number of points they have shot.) The number of points ranges from 0 (the arrow is outside the white rings) to 10. See the image below.



Generate archers and scores

For generating a list of archers and their scores a so-called factory method, `generateArchers`, is available in the Archer class. A factory method is a method that

takes care of all the details of construction complex instances of classes. In this case the factory method takes care of generating a number of archers, assigning each archer a first name, a surname and lets them should 30 arrows. After each round of 3 arrows judges report the scores to each archer, so when the list of archers is returned by the factory method, all archers have been notified of their scores.
In case you don't want to use the factory method (for example in your unit tests) you must ensure that the code that creates the instances resides in the same package as the `Archer` class. (In other words, you are NOT allowed to change the visibility of the constructor(s)!)

The judges use the method `registersScoreForRound` to report the score to an archer. For more information read the JavaDoc of this method.

If you want to create archers for your unit-tests with known ID's, you can use the constructor that takes three parameters. **Don't** use this constructor for anything else but unit-tests!

Missing code

Here you find some information on what you must do. There is also valuable information to be found in the JavaDoc of the provided classes.

The `toString()` method of Archer should return a `String` in the following format: "135787 (225) Nico Tromp", so first the id, then between brackets the total number of points followed by first name and surname separated by spaces.

The missing pieces of code that you still need to implement are:

- Ensure that the ID can never change once it has been assigned a number
- Ensure that each archer gets a unique ID that increases by one for each new archer (the first archer created has ID 135788), except when using the constructor with three parameters.
- Add code to store the scores, the method `registerScoresForRound` should be used for that. Think of a nice data structure that can be used to store these scores. Should it be a 2-dimensional array, a list of arrays, a list of lists of int's, or …. **IMPORTANT**: please be aware of the fact that this method uses 1 for round 1, 2 for round 2 etc.., in contrast to array's which are 0-based!

TIPS:

- The provided code is not completed yet.
- It is perfectly ok to create attributes if you find it useful to have them. Please add an explanation to your report in which you make it very clear why you added these attributes!
- It can be helpful if you create a class (can be a plain Java class with a `main` method, or a Junit test class) that requests a list with a small number of archers from the factory-method and then loop over the archers and print each and every one of them. Then you can figure out which code is already there and which is missing.
- You can force an archer to have a certain score by defining an `int` array, fill it with some nice values, and call for each of the rounds the `registerScoreForRound` method with that predefined score for an archer or archers. So it is possible to

override[1] the score that the judge gave the archer. By doing this you can know beforehand what the output should look like. And of-course you can use the three-parameter constructor as well.

Finding the champion

Now that you are able to generate lots of archers and their scores, it is time to determine who is the champion.

Scoring

As mentioned before depending on the exact competition the scoring can be different. That is the reason why your application must be able to handle different scoring schemes. For now you can focus on the following.

1. When comparing the total number of points between two archers the one with the most points 'wins'.
2. If the number of total points is equal, the archer with the most 10's wins. If the number of 10's are equal the archer with the most 9's wins.
3. If it is still a tie, then the less experienced archer wins. (Experienced archers have a lower ID then less experienced archers.)

Now that you know how to determine the champion it is time to find the most efficient way for determining the champion. You need to sort the archers in such a way that the champion is the first in the list and the archer with the lowest score is at the end.

Efficiency

For finding the most efficient sorting algorithm you must compare three following sorting algorithms:

1. Either selection-sort or insertion-sort
2. Quick-sort
3. The sort method provided by Java for Collections (Usually a matter of calling the `sort(Comparator<E>)` method on the collection)

For the first two algorithms you have to write your own implementation.
Now you have to determine how long it takes for each algorithm to sort a list of archers and their scores. You start with 100 archers and multiply it by 2 and keep doing this until you reach 5.000.000 archers or until sorting the list takes more than 20 seconds. *When measuring the time it takes to sort the archers, make sure you only measure the time needed for actually sorting the archers, and not for creating them!*
To ensure that your comparisons are fair it is important that you use the same unordered list of archers for each algorithm. So once you have a list of archers make three copies of it and use each copy for a different algorithm.
Since your computer has more tasks to perform (playing songs through Spotify, scanning files for viruses) than only determine the champion for the competition, you must run each experiment 10 times and use the average over these 10 runs as input for the determination of the efficiency.

[1] This 'override' has nothing to do with overriding a method!

If you run the tests with the JVM parameter `-Xint` you will find that the results are different than without the parameter. Include an explanation for phenomena in your report.

Now that you have the numbers you need to determine (by using math) the efficiency of each algorithm.

Report/grading

If your report meets the following criteria your score can be at best 'Good':

1. It must follow the guidelines as described in the document 'Report-Requirements.pdf'
2. You have ensured by using the correct Java keyword that the ID of the archer can never be changed (see JavaDoc).
3. You must handle the fact that rounds are counted starting from 1 efficiently. That means that you are not allowed to claim more memory that needed for registering 30 scores per archers.
4. If you added one or more constructors to the Archer class your report must contain a separate section describing the reason why you needed that constructor(s).
5. For any method that you added to any class (excluding simple getters and setters) please explain what the purpose of the method is and explain how it works.
6. Your report contains all the timing results.
7. The correct efficiency has been determined for each of the sorting algorithms using mathematical constructs.
8. Your report contains a graphical representation of the efficiency.
9. You included an explanation for the difference when running the experiments with and without the JVM parameter `-Xint`.
10. The provided code contains some unit tests but they are by no means complete. You have to add tests which will show that your code works as expected. Any test that you create can be added to the existing test classes.

Optional

If you implemented any of the optional code and the grade so far is 'Sufficient' or better AND your code is able to pass the following criteria, then your grade is moved one step towards 'Excellent'.

1. Your code passes all the unit tests.
2. Your report contains a clear and human readable explanation on how the code works. In other words, you must explain Quick-sort in layman's terms. Failing to do so will not get you the extra points even if the code works!
3. Your code passes extra tests. These extra tests are not ,and will not made publicly available…