Genetic algorithms

For this assessment I ask you to optimise two functions (problems) using genetic algorithms. The assessment will be partially marked by machine. It is therefore very important that you follow the instructions given to you very carefully. Failure to do so will result in substantial loss of marks.

To help you complete the assessment you are given 2 pieces of software: (1) A a shell program code called Example.java. (2) A compiled class file `Assess.class` that implements the problems. It contains 2 methods, namely `double Assess.getTest1(double[])` and `double[] Assess.getTest2(boolean[])`. These functions accept candidate solutions for problems 1 and 2 respectively and return the fitness of the proposed solutions. The example code in `Example.java` demonstrates how to access them. Look at the code carefully to understand how it works. Additionally, it also contains a function `checkIn(...)`. You will need to call this function at the end of your assignment in order to indicate the solutions to the two problems given to you. See below for more details.

Problem 1

The first problem is a black box function that takes 20 real-valued numbers as input. The function has a unique minimum fitness of 0 for a specific input. Your task is to write a genetic algorithm to find this minimum value. You can safely assume that the minimum can be found for input values between -5.0 and +5.0 for all variables/dials. So, you do not need to search smaller or larger values than that.

Problem 2

The second problem is a version of the well-known knapsack problem. The idea is that you want to pack a bag. You have 100 items that you would like to take with you. Each item has a utility value and a weight (assumed to be expressed in kg). You want to maximise your utility (which is the sum of the utilities of all the items you take with you) but you are not allowed to pack more than 500 kgs of weight altogether. This means that you need to choose the items that maximise your total utility while staying within the weight limit.

The knapsack problem is already implemented for you, so you do not need to worry about the list of items and their utilities of weight. The function

```
double[] Assess.getTest2(boolean[])
```

accepts a boolean array of length 100 as a candidate solution. The item you want to take with you corresponds to the index value and the boolean at the relevant index encodes whether or not you take the item with you. So, for example, if the array is given by:

```
true,true,false,true,false,false,....
```

Then this means that you take with you items 0,1,3,...

The function `getTest2(boolean[])` returns an array of doubles. The first entry (index 0) is the weight of your combination and the second (index 1) is the sum of the utilities. If unsure, check the example code where I show how to use the function. Before starting to implement your GA, play a bit with the example code to make sure you understand how fitness is returned.

Your task is to find the a combination of items to pack that maximises utility while staying within the weight constraints.

Marking criteria

Your final submission will be assessed by compiling and running it on raptor using the above two commands. It is your responsibility to ensure before submission that they work and produce the desired result. **If this does not work, you will be awarded 0 points.**

Non-compiling/non-running programs will get 0 marks.

Rules on use of APIs: `Example.java` includes, at the moment

```
java.lang.Math
```

and

```
java.util.*
```

These allow you to use mathematical functions in java and ArrayLists. You are not allowed to use any APIs beyond that. In particular, do not use any graphics or GUI. This is not necessary and not allowed.

- You will get up to 40 marks for a program that compiles, runs, displays functionality and completes within 20 seconds on raptor.
- You will get up to 40 additional points for the first task. The number of points will depend on the quality of the solution. The closer the solution fitness to 0 the higher the mark. Note that I am asking for the perfect solution. Do not assume that a fitness of 0.00000000000001 gets full marks, but attempt to find the 0 fitness solution. That said, you will get partial marks for good solutions.
- You will get up to 20 points for the second problem. The higher the utility (while staying within the weight constraints) the more points. If your solution is above the weight limit (even by a little bit), it will not get any extra points.
- Your program must completed within 20 seconds, for both tasks. The time will be measured on raptor. Note that if you have a very slow or very fast machine at home, execution times can be substantially different. So, make sure that you check the run-time on raptor.
- If your code takes longer than 20 seconds there will be substantial point deductions.
- After 60s I will terminate the execution and your marks are capped at 40.