The Garage

Submission deadline	**Thursday, 19 June 2014, 17:00**

Task

Background:
You are the proud owner and the only employee of Auto Repair Inc. When you started the company, you chose not to charge by the hour, but based on the complexity of the repairs. Some repairs are really complex but take very little time. Others may require hours, but are so simple that their list price is low.

At the moment, business is great. There are more broken cars than you could possibly repair. You want to generate as much revenue as possible, so you'll need to decide which cars to repair.

Instructions:
Your task is to write a class that implements the following interface:

```
public interface Garage {
    /**
     * Takes a set of Car objects as a parameter, and returns the subset that
     * can be fixed in the given timeframe while generating as much revenue
     * as possible.
     *
     * @param cars Set of cars waiting to be repaired
     * @param availableHours The number of hours you have to repair the cars
     * @return Set of cars that have been repaired
     */
    public Set<Car> repair(Set<Car> cars, int availableHours);
}
```

You'll also need to implement the Car class. You can implement it in any way you like, but it must include the following getter and two fields:

```
/**
 * @return The registration number of the car
 */
public String getRegistrationNumber();

/**
 * The number of hours it takes to fix the car
 */
```

```
private int repairHours;

/**
 * The number of euros you can charge the customer for the repairs
 */
private int repairPrice;
```

For the purpose of the assignment, each repair consumes only your time - you can, for example, ignore the cost of spare parts.

Example Data

Available hours: *8*

Cars:
ABC-123 3h 100€
DEF-456 5h 120€
GHI-789 4h 80€
ZZZ-999 1h 50€

Result: ABC-123, GHI-789, ZZZ-999

The solution for knapsack problem can be applied here for getting the maximum revenue. The 0/1 Kanpsack problem can be defined as such:

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit (capacity of knapsack) and the total value is as large as possible. Dynamic programming could be used.

You are asked to finish the following tasks:

1. Design a Car class describing the states of a car, for example, repairHours, repairPrice;
2. Design a RepairCar class for getting maximum revenue for an auto repair company, which implements the Garage interface;
3. You can solve the maximum revenue using the solution of knapsack problem;
4. Design a simple GUI by Java Swing to test your implementation.

Marking Scheme (100 marks)

20 Marks Car class

To gain 20 marks, your Car class has to represent the car information correctly.

20 Marks RepairCar class

To gain 20 marks, your RepairCar class has to get the maximum revenue from the test data.

20 Marks Knapsack problem

To gain 20 marks, you have to learn and implement knapsack problem.

20 Marks GUI design

To gain 20 marks, your application needs to have a GUI to input the data and output the result.

20 Marks Testing and report

To gain 20 marks, the program has to work correctly and print the results as expected. A report on the task is also necessary.

Correctness, robustness, error-handling, proper functionality and documentation will be assessed for these areas where appropriate. The following is a guide for the marking:

- First (>= 70 to 100 marks): Correct code implementing all the functionality, is neatly formatted and clearly commented with few errors and a well written and structured report showing a good grasp of programming issues for this assignment.

- Second Upper (>= 60 to 69 marks): Working code implements all the functionality required with limited errors, generally good layout and comments in the code, and a report showing a good understanding of programming issues for this assignment.

- Second Lower (>= 50 to 59 marks): The code implements most of the required functionality with minor errors, some comments and a report showing at some understanding of programming issues for this assignment.

- Third (>= 40 to 49 marks): Essential functionality designed and implemented with some errors, with some comments and a written report describing some of the work done.

  Fail (<= 39 marks): Not satisfy the pass criteria and will still get some marks in most cases.

- None-submission: A mark of 0 will be awarded.

Plagiarism and Collusion

This assignment is individual work. Cheating will not be tolerated and will be dealt with in accordance with the University Code of Practice on Assessment. Working with each other on homework and lab assignments is encouraged and expected in this course, as long as it leads to independence and greater learning. However, copy of any material submitted for grading will not be tolerated. Also, over-reliance on the help of another student to the point of dependency is unhealthy and goes against the spirit of the honour code.

Submission of Work

You are required to submit the following items **by the deadline**:

A documentation on your implementation and the source code files of your implementation zipped into a single file (so that your source code programs can be run by the marker if needed) **via server**. Your submission needs to contain the information of your name, ID number, University e-mail address.