Last updated: 18 November 2020

Maynooth University
National University of Ireland Maynooth

ASSIGNMENT DESCRIPTOR

Programme (Level):	**MSc**
Module:	CS613 – Advanced Concepts in Object Oriented Programming
Assignment Number:	Assignment 01
Date of Title Issue:	18th November 2020 (Wed)
Assignment Deadline:	16th December 2020 (Wed - 7pm) (~4 weeks to Complete)
Assignment Submission:	**see below (Groups necessary: 3 persons – exceptional sizes (2, 4, 5) only if granted.**
Assignment Weighting:	**20% (of overall Module)**

Assignment Title: GA Structure using Singleton, Strategy and Factory Design Patterns

Learning Outcomes

- Construct object oriented programming solutions for reuse, using abstract data types that incorporate encapsulation, data abstraction, information hiding, and the separation of behaviour and implementation.
- Construct multiple-file or multiple-module programming solutions that use classes, subclasses, class hierarchies, inheritance, parameterised classes, generics and polymorphism to reuse existing design and code.
- Create programming solutions that use data structures and existing libraries.
- Perform object-oriented design and programming with a high level of proficiency ~~in more than one object-oriented programming language.~~ *(Only a single programming language required for this assignment)*

Submission

1. Create the following folder structure:

```
1234567_CS613_GAPatterns20pc
        ├──bin
        └──src
        └──doc
        └──README.txt
```

Only Put your actual StudentID instead of "1234567"

- The source-code for your submission (`.java`) files should be stored in the …`\src` directory and the byte-code (`.class`) files should be stored in the respective …`\bin` directory.
- A `README.txt` file must be submitted which includes instructions of how to compile-and-run the code, as well as a brief explanation of the output and what it shows. There must also be a section with sub-titles (e.g. __singleton pattern__, __strategy pattern__, __factory pattern__) where under each section, it is indicated where, in the code, i.e. which .java files - each pattern can be found and the variety of pattern implemented and why this was chosen where applicable.
- The …`\doc` folder should contain `javadoc` documentation of your classes where appropriate.

What do I have to do?

Apply the Singleton, Strategy, and Factory patterns a Genetic Algorithm (GA) problem as described below.

NOTE(Important): Graded marks depending on your Factory pattern implementation:

If you implement an **"Abstract Factory"** pattern you can achieve **100 marks**.

If you implement a **"Factory method"** pattern you can achieve a **maximum of 95 marks**.

If you implement only a **"Simple Factory"** pattern you can achieve a **maximum of 90 marks**.

- **Research Genetic Algorithms (GAs) and Human Evolution (Recommended no more than 4 - 6 Hrs recommended/needed).**

  Source for Genetic Algorithm Explanation: http://www.obitko.com/tutorials/genetic-algorithms/

  Source for Human Genetics Explanations: https://www.23andme.com/gen101/genes/

  NOTE (Important): You are **not** required to develop a fully functioning genetic-algorithm. You are required to get a basic understanding of genetics, and the genetic-algorithm concepts of: Selection, Reproduction (/Crossover) and Mutation. You are only **required** to develop a proof-of-concept solution.

  Feel free to find your own resources or do your own research or delve deeply into the problem if you wish: however, **focus on the marking scheme** below first, **as this outlines what work will produce marks**.

- **Develop a prototype Genetic Algorithm (GA) that uses the Singleton, Strategy and Factory Design patterns.**

  Your prototype GA code should use the Strategy design pattern to allow different GA Selection, Reproduction (/Crossover) or Mutation implementations (or strategies) to be dynamically changed at runtime.

  Your prototype GA code should select an appropriate Object to demonstrate use of a singleton Pattern. (Full marks for comments and implementation that is thread-safe and an explanation of same).

  Your prototype GA code should use some variant of a Factory design pattern, to handle the instantiation (creation) of the different Selection, Reproduction (/Crossover) or Mutation objects.

  (Again, **you only need a proof-of-concept solution,** which executes to demonstrate how the three patterns could be applied to a GA design; not a fully functional GA.)

Resources/Research:

Books (Java): Freeman, E., Freeman, E., Sierra, K., Bates, B., 2004. *Head first design patterns*. O'Reilly.

Strategy Chapter: "Intro to Design Patterns" (Chapter 1). pp. 1 – 36.
Factory Chapter: "The Factory Pattern" (Chapter 4). pp. 109 – 169.
Singleton Chapter: "The Singleton Pattern" (Chapter 5). pp. 169 – 190.

Books (C++): Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. *Design patterns : elements of reusable object-oriented software*, Addison Wesley, Reading, Mass.
Factory Chapter: "Abstract Factory" , "Factory Method". {pp. 87 – 97, 107 – 117}.
Singleton Chapter: "Singleton". pp. 127 – 135.
Strategy Chapter: "Strategy". pp. 315 – 325.

Web (Patterns):
https://en.wikipedia.org/wiki/Singleton_pattern
http://en.wikipedia.org/wiki/Strategy_pattern
http://en.wikipedia.org/wiki/Factory_method_pattern

Marking Scheme (How will marks be awarded to my work?)

Clarity, cleanliness of code (/**including code documentation*/), and submitted exactly as requested including README.txt and \doc folder to document the code (e.g. Javadoc tool)

(10 marks)

Data abstraction **used appropriately throughout**

(05 marks)

Information hiding **present and appropriate throughout**

(05 marks)

Separation of behavior and implementation **particularly with respect to the Strategy & Factory design patterns.**

(05 marks)

Multiple-file or multiple-module programming solution submitted - classes, subclasses (class hierarchies/inheritance), generics and polymorphism used appropriately to reuse existing design and code throughout.

(05 marks)

Programming solution submitted creates data structures that can be objectively judged to be cohesive (facilitate maintenance), facilitate reuse/extensibility, adhere to the principle of substitutability and the open-closed principle – and reuse existing libraries **where appropriate.**

(20 marks)

Singleton pattern correctly applied to allow a single instance of a class to be created only.

(10 marks)

Strategy pattern correctly applied to allow dynamic change of behaviour at runtime

(15 marks)

Factory pattern correctly applied to handle instantiation of GA operator objects.

(25 marks)

Total Marks (100 marks)