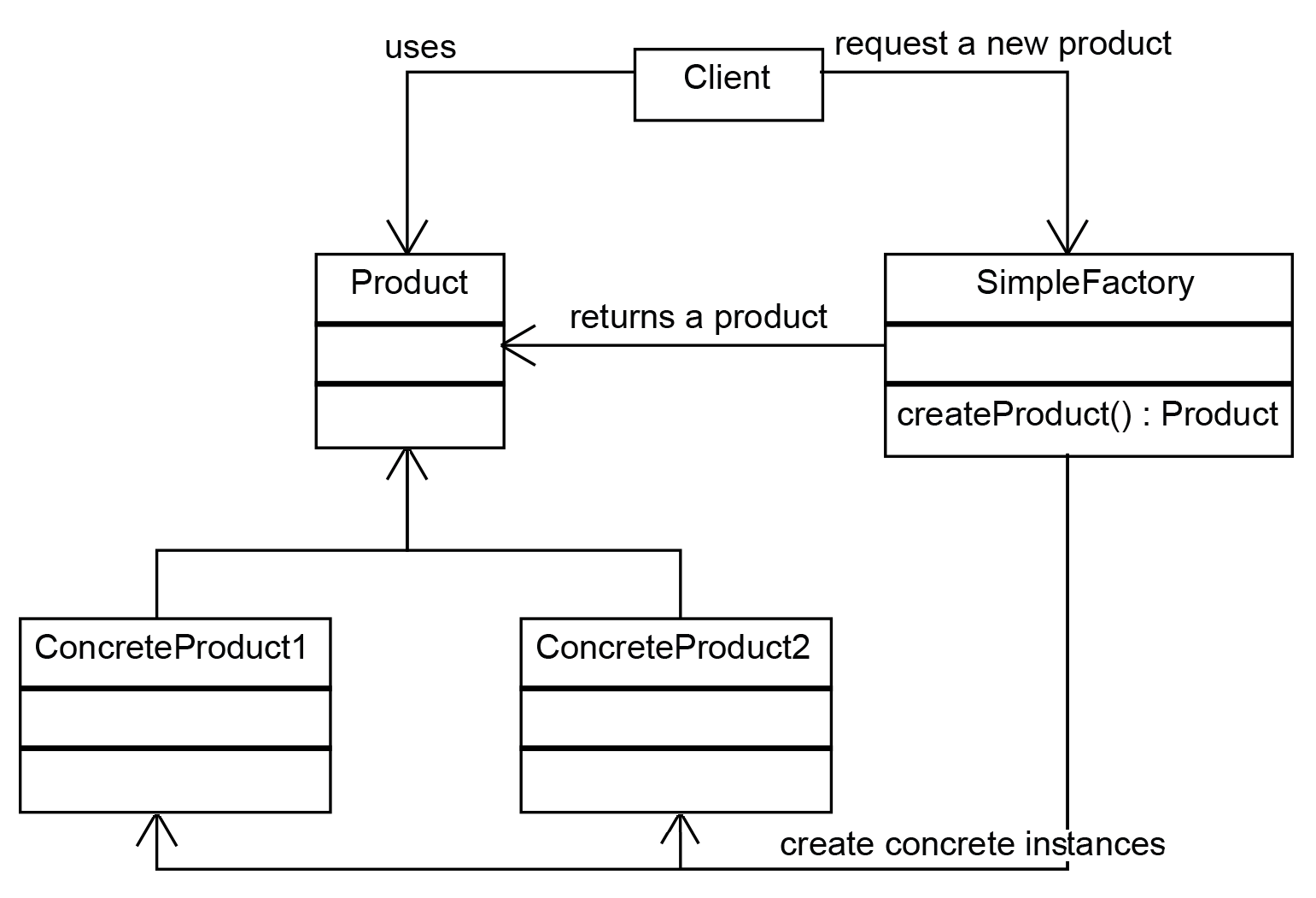
**Background Information**

Simulations of imagined or real situations in nature are frequently created to provide an experimental environment for studies as wide ranging as astronomical events, battlefield encounters, and the clustering mentality of insects and birds. These types of software are most efficient when designed following an object-oriented paradigm.

**Your Assignment**

You are to create a program that simulates the behavior of Digital One-Celled Organisms (DOCOs) within a simulated world. In this programming assignment the DOCO behavior will be modified and expanded from that in programming assignment 1. The basic requirements for this assignment are listed below. All of the original requirements of programming assignment 1 are still in effect unless modified in the new requirements

Version 2.0 of this software must incorporate the Strategy, Abstract Factory, and Singleton design patterns in appropriate ways. The program design **must** conform to the principle of **loose coupling** where ever possible.

1. In this version of the software, DOCOs will randomly move from their current location to an adjoining grid cell. There will be three possible patterns of movement. Each DOCO will exhibit only one movement pattern. The following conditions apply.
   * Behavior pattern 1 will cause the DOCO to move **only** in a horizontal direction. If an edge of the world is encountered the DOCO will randomly elect to move up or down a row and reverse its direction of movement.
   * Behavior pattern 2 will cause the DOCO to move **only** in a vertical direction. If an edge of the world is encountered the DOCO will randomly elect to move left or right a column and reverse its direction of movement.
   * Behavior pattern 3 will cause the DOCO to move **only** in a diagonal direction. If an edge of the world is encountered the DOCO will randomly elect to move left, right, up, or down and either reverse its direction of movement or move in the other diagonal direction.
   * There must be a class added to the application to handle each of the required behaviors. These classes must conform to the **Strategy** design pattern.
   * Which behavior pattern a DOCO exhibits will be provided in the data file used to set up the simulation. A modified DataParser will be provided.
   * Before the simulation starts each DOCO will randomly select a starting direction to move. This may be to the North, Northeast, East, Southeast, South, Southwest, West, or Northwest. The starting direction shall be determined within the constraints of its defined movement behavior pattern.
   * Each DOCO will always move in a straight line in its' current direction of movement unless that movement is modified by the constraints given below.
   * If there is another DOCO in the next cell along its' current heading then the DOCO will reverse direction. The new heading, however, must not take it into the cell of another DOCO. If there is a DOCO on either side of its current direction the DOCO will move to a different row or column according to its defined behavior pattern, e.g. a DOCO with horizontal movement behavior will move up or down a row.
   * If a DOCO "smells" a food pellet in a cell bordering its' current location it will alter its heading to take it into that cell on the next move. The cell containing food does not have to be along its' path defined by its movement behavior.
   * If a DOCO "sees" an obstacle in the next cell along its current direction it will modify its movement to avoid that obstacle.
2. A new class, to be named **DOCOFactory** is to be added to the application. It will follow a modification of the **Abstract Factory** which is sometimes referred to as a *Simple Factory*. The **DOCOFactory** class will also be implemented following the **Singleton** design pattern. Its purpose will be to create DOCOs of the required type. (We will discuss this in class.)
   * 
   * <https://airbrake.io/blog/design-patterns/software-design-patterns-guide>
3. DOCOs maintain an internal energy level that is initialized to 500. Each time a DOCO moves from one cell to another it expends 25 units of energy. This is a change from program 1. (Note: additional behaviors that expend energy may be added later.)
4. A predefined number of food pellets will be introduced into the world initially at random locations. At one second intervals a random number of additional food pellets (between 1 and 10) will be added to the world and placed at random locations. At most three food pellets can be placed in a single cell.
5. If a DOCO enters a cell containing food pellets it will eat all of the pellets in the cell and gain 50 units of energy for each pellet consumed.
6. If a DOCO's energy level drops to zero it dies and disappears from the world. This means it is removed from the collection of DOCOs in the simulation and deleted.
7. If a DOCO's energy level goes up to or above 750 it will divide into two identical DOCOs each having half of the energy units of the parent cell and in the next interval each will move off in different random directions following the inherited behavior pattern of the parent cell. (Note: This implies that, at least momentarily, two DOCOs will occupy a single cell.)
8. A predefined number of obstacles will be placed in the world at locations defined in the new data file. A DOCO may not enter a cell defined as an obstacle.
9. At one second intervals a diagram will be printed on the screen similar to the one used in programming assignment 1. The symbols are as follows: '=' = location of a DOCO exhibiting horizontal only movement, '|' = location of a DOCO exhibiting vertical only movement, 'X' = location of a DOCO exhibiting diagonal only movement, '.' = one or more food pellets, '-' = an empty cell, and the ASCII character with the hex value 0xB2 = an obstacle. Note: You can print this character with code like this: **cout << "\xB2";**

This project shall be created as a Win-32 Console Application. It shall **NOT** use precompiled headers.

**Deliverables**

Schedule information:

|  |  |  |
| --- | --- | --- |
| **Task** | **Schedule** | **Fall 2020 Dates**  **(Not Official)** |
| Release assignments | Week 0 | 10/11/2020 |
| Class Outline | Week 2 | 10/25/2020 |
| Functional Outline | Week 3 | 11/01/2020 |
| VS Project | Week 5 | 11/15/2020 |

A short description of the required deliverables is given below. Additional requirements may be provided in the Canvas assignment.

These products as specified below shall be uploaded to Canvas for the applicable assignment. **The due dates in Canvas are always the official due dates/times.**

**Preliminary Class Diagram --** The class diagram shall be drawn using standard UML notation and shall show all of the classes to be implemented in the software and their relationships (dependencies, associations, generalizations, realizations, etc.)

**Class Outline --** The class outline shall list all proposed variables and functions in each proposed class with a brief description of what each does.

**Functionality Outline --** The functionality outline shall be a pseudocode outline which will show the step-by-step functionality of each function in each class of the program. This should be taken out to a fair amount of detail.

**Final Project --** The entire software project (the solution folder not just the project folder) compatible with Microsoft Visual Studio 2017 shall be compressed into a zip file and submitted for grading.

**Supplemental Material**

Students are allowed the following material that can be accessed by Canvas:

* A new Data file DOCOData02.xml

**Assignment Information**

This program has been assigned for the following terms:

Fall 2020