Abstract

We propose an object-oriented programming library system for Science in a similar structure of .NET framework. We call the system Science Code .Net. Presenting rules of thumb, we show how to enhance readability of Science Code .Net. As an initial progress, several namespaces have been constructed. We explain the structure of classes in each namespace. As a challenge to the programming community, a perspective view of Science Code .Net is discussed.

Keywords: Object-oriented programming; .NET; C#; Framework design; Numerical recipes; Physical sciences

1. Introduction

Reusability of source codes is one of the main issues in the field of computer software, and even in scientific computing. The textbook named Numerical Recipes [1] provides reusable source codes for scientific computing. Computer Physics Communications (CPC) Program Library [2] was established in 1969 in order to accumulate programs in Computational Physics and Chemistry, and now contains several thousands of programs. The CPC Library classifies all accumulated programs according to subject. However, this simple accumulation has a weak point that the source codes are not correlated, and are written for case-by-case. In this approach, library source codes and driver source codes are intermixed, so that it is not easy to modify source codes when we encounter slightly different problems. As for reusability, the web-based program library called netlib [3] is most commonly used. An advantage of netlib is that there are editors for each computational methodology. These editors control the evolution of the code. Still, we find that, in scientific computing, netlib does not provide us with clear and easy-to-use connections between methodologies and physical subjects. For these reasons, both the libraries are not sufficient to satisfy the scientific community not only in contents-wise but also in concepts-wise. Thus, it is necessary to develop a source code library system based on a new concept, which combines good points from both libraries.

Recently Microsoft introduced .NET technology [4] to improve the reusability of code. As a remarkable software framework, .NET organizes all codes of about 2500 classes according to readable namespaces. Programmers for .NET carefully choose names of classes and methods in order to simplify access.
The main purpose of framework is to provide “Lego” pieces for rapid construction of new codes. It is important to teach how to make the “Lego” pieces [5]. Although the best examples of “Lego” pieces are .NET framework classes, the source codes are hidden. Thus, as far as education for framework design is concerned, .NET itself is not enough. Classic problems, such as well-known algorithms and data structures, are used as good sources for teaching Java framework design [6]. We find a good explanation on object-oriented algorithm analysis and design with Java [7]. However, these limited frameworks are related with only Computer Science, whereas the concept of framework should be understood by all students in different areas. In fact, we are forced to build up software frameworks for other areas beyond Computer Science. Physical Sciences are adequate areas, to which framework design is applied. For several specific subjects in Physics, we list some approaches of framework design: PIC (Particle in Cell) [5], RunMC (Monte Carlo of high energy particle collisions) [8], Advection-diffusion model [9], Time-dependent Maxwell equations [10]. However, the scopes of these frameworks are limited. We need a new fresh start to integrate software frameworks for the whole Science. Since students may change their majors from Physics to Chemistry or from Chemistry to Biology, we should design a consistent framework for easy transition or integration of knowledge. Hence, our perspective framework should contain all concepts of Mathematics, Physics, Biology, Chemistry, and Statistics in which all classes are written in a coherent fashion.

The author [11] proposed that the strategy of the .NET framework is applicable to scientific computing. The main difference of this proposed approach from the simple accumulation of CPC is that all source codes in our system should be organized like .NET. The purpose of this paper is to present rules of thumb for a refined structure of a framework and to address a challenge to the programming community. All communities for the scientific computing are invited to participate in order to build up a software framework which is scalable and self-organizable.

We now call our perspective software framework **Science Code .Net**. For **Science Code .Net** to be self-organizable, we organize subdirectories of **Science** by using titles of textbooks. As books in a library are classified according to subject, we organize subdirectories. The following is a simple reference for subdirectories as a suggestion.

- **Science**
  - Biology
    + GeneralBiology
    + MolecularBiology
    + PopulationGenetics
  - Chemistry
    + AnalyticalChemistry
    + Biochemistry
    + GeneralChemistry
    + OrganicChemistry
    + PhysicalChemistry
    + PolymerChemistry
  - Mathematics
    + Calculus
    + Graphics
    + IntegralEquation
    + LinearAlgebra
    + OrdinaryDifferentialEquation
    + PartialDifferentialEquation
    + SpecialFunction
  - Physics
    + ClassicalMechanics
    + Electromagnetism
    + GeneralPhysics
    + QuantumMechanics
    + StatisticalMechanics
  - Statistics
    + Regression
This classification is inspired by the namespaces of Microsoft .NET framework class library. We believe that more subjects can be added or adjusted by future needs from the corresponding community.

In each namespace, classes are named after each item of the index in textbooks. Use of the same names in classes or methods is expected to make it easier to understand source codes, because students are already familiar with the terminology in Science.

Unlike .NET, some classes in Science appear in multiple places. For instance, class Vector appears in both GeneralPhysics and Calculus. This multiple appearance is required since all classes in each subject are self-contained. The detailed discussion is given in the next section.

We note that coding Science is a huge work, and we will never see the end of it. In this paper, we introduce the start for the work of building up Science. We first describe rules imposed in order to increase readability of source codes, and then show some parts of Science which would be useful in educational courses for Mathematics, Physics, and .NET framework.

2. Rules of thumb

There are two types of errors occurring in programming. One is syntax error and the other is logical error. A computer can locate syntax errors and inform programmers directly. However, logical errors should be resolved by programmers.

To reduce logical errors, it is useful to make additional rules other than computer language grammars. These rules not only reduce the possibility of logical errors but also increase readability.

In order to increase reusability, we propose a rule, which is described by a single statement: "physical inputs and outputs in a self-contained structure". Three folds of this rule are explained in detail as follows.

(1) Give full names for outputs. When we use words as names for public constructors, properties, methods, or delegates, the words must be found in a dictionary. We use uppercase characters to combine two or more words like QuantumMechanics or FermionOneBodyHamiltonian. Uncommon words, like amoeba or vegas in Numerical Recipes, must be classified as names of private methods.

(2) Use abstract data type for inputs. For parameters of public constructors, methods, and delegates, any primitive data types like double and int are not allowed. For instance, one may write a constructor for class Hamiltonian such as

\[
\text{public Hamiltonian(double coupling)}\{\ldots\}
\]

Then, calling Hamiltonian, one may write a statement like

\[
\text{Hamiltonian } h = \text{new Hamiltonian(0.3)};
\]

We note that it is not straightforward to understand what the number 0.3 means unless comments are made. If we follow the above imposed restriction that all inputs are of class types, the above constructor should be rewritten as

\[
\text{public Hamiltonian(Coupling c)}\{\ldots\}
\]

For the model-dependent part, we may write statements as follows,

\[
\begin{align*}
\text{Coupling } a &= \text{new Coupling();} \\
\text{a.Strength} &= 0.3; \\
\text{Hamiltonian } h &= \text{new Hamiltonian(a);}
\end{align*}
\]
Although we should make another class `Coupling` and write more lines in the model-dependent part, this rule of physical inputs will increase readability in a large project. It should be emphasized that public constructors and public methods need comply to this rule. We can freely use any primitive types of inputs for private methods in a class. Furthermore, for the classes, which are under `Science.Mathematics`, we remove this restriction of abstract data type inputs, because a number itself becomes a concrete object in Mathematics.

(3) **All related classes must be in a single directory.** According to the second rule, we should make multiple classes in turns. For the above example, we should make the class `Coupling` for the class `Hamiltonian`. `Coupling` may need another class which is one of the input parameters. We keep the process of making classes. However, this process should end eventually. The third rule is that in this process all related classes should be in a single directory for easy access and handling. If related classes are located in different directories or fields, then it is impossible for each field to evolve separately. Thus, whenever we need a class, which is written previously in other namespace, we simply copy the class into our current directory and modify it. For example, when we need class `Vector` in `GeneralPhysics`, we try to find the class in `Calculus`, and we copy and add it into `GeneralPhysics`. In consequence, all source codes for a subject should be self-contained in a single directory. As another example, we present the part of constructor in `AngularMomentum` as

```java
public AngularMomentum(Position r, Momentum p){...}
```

We notice that the rule of physical inputs is satisfied. In order to satisfy the self-contained structure, we should make classes: `Position` and `Momentum`. For `Momentum`, we make `Mass` and `Velocity`. Finally, for `Velocity`, we make `Length` and `Time`. Then, all these classes should be in the single directory of `GeneralPhysics`. Sometimes, in order to avoid heavy copying source codes, one can take an exception to the rule of the self-contained structure.

With the three rules suggested above, the library system for education `Science` is initiated. We hope that introduction to Calculus with `Science.Mathematics` is more fruitful than with commercial softwares such as Mathematica, Maple and Matlab. It is recommendable to use a free software to compile C# files. We note that #Develop [12] is a free IDE for C# projects on Microsoft .NET 2.0 platform.

### 3. Initial progress

At present, small parts of `Science` including `Biology`, `Mathematics` and `Physics` have been constructed. The library of `PopulationGenetics`, `Calculus`, `LinearAlgebra`, `OrdinaryDifferentialEquation`, and `GeneralPhysics` is now ready to be used. All source codes can be downloaded from the web site: [http://www.ScienceCode.net](http://www.ScienceCode.net).

We present class names in 5 subdirectories of `Science`.

```text
@ Science.Biology.PopulationGenetics
 - Chromosome.cs
 - FemaleGenotype.cs
 - Gene.cs
 - MalePhenotype.cs
 - Mitochondria.cs
 - Reproduction.cs
 - SocialGene.cs
@ Science.Mathematics.Calculus
 - Complex.cs
 - Derivative.cs
 - Vector.cs
@ Science.Mathematics.LinearAlgebra
 - Complex.cs
 - Function.cs
 - IdentityMatrix.cs
 - Matrix.cs
```

We use the textbooks written by S. Lang for Calculus [13] and Linear Algebra [14]. Using a standard textbook, we made 119 explicit class names from Acceleration to Work for General Physics [15]. These class names can be found in the index of any corresponding textbooks. The source code for these classes can be seen on the Internet. We should remove or add some entries in the future.

For Linear Algebra, it is reasonable to consider inheritance for classes related with matrices. We present the hierarchical structure of matrix as shown below:

\[
\text{Matrix} \downarrow \text{SquareMatrix} \downarrow \text{IdentityMatrix, DiagonalMatrix, RealSymmetricMatrix, HermitianMatrix}
\]

We find that the key property names for Matrix, SquareMatrix and Hermitian Matrix are Transpose, InverseMatrix and Eigenvectors, respectively.

Since all physical quantities have physical dimensions, the concept of dimension appears first in General Physics. We build the hierarchical structure between classes in General Physics as follows. The uppermost position is occupied by Dimension. Two classes, Scalar and Vector, are derived from Dimension. All other classes for physical quantities inherit properties and methods from either Scalar or Vector. Thus, the hierarchical structure of physical quantity classes is shown as below:

\[
\text{Dimension} \leftarrow \text{Scalar} \downarrow \text{Work, [and 33 entries]} \leftarrow \text{Vector} \downarrow \text{Force, [and 21 entries]} \downarrow \text{ElectricForce, LorentzForce, MagneticForce}
\]

The key property of each class is given by the corresponding unit of physical quantity. For instance, we use \( \text{m} \) for class Length, \( \text{kg} \) for Mass, and \( \text{s} \) for Time. Similarly, we make the property \( \text{N} \) (Newton) for Force, and \( \text{J} \) (Joule) for Work and Kinetic Energy.


These organized source codes in Science should mirror the concepts of Science. A future student, who is interested in Science, should learn parallel frameworks of Science Code.Net. In consequence, the student will become a future author, who will make a new library system for his own subject like QuantumMonteCarlo, and will add it into Science Code.Net.
4. Conclusion

Being inspired by .NET, we have proposed a library system for Science. It is emphasized that class names should be the same as index items in textbooks. Furthermore, namespace names are determined by titles of textbooks. When new subjects appear in new books, we can add new namespaces into Science.

In Science, the subjects are so diverse that many authors should cooperate to build up Science Code .Net. If we allow authors to use directly classes belonging to a different namespace, then conflicts will appear when other authors modify those classes to evolve. Thus, as genes in the real world are duplicated from a cell to another cell in order to place information nearby, we copy classes in other subjects into our new subject. We modify the copied source codes so that they are slightly different from each other in many namespaces. For example, class Force would appear in both Science.Biology.PhysicalBiology and Science.Physics.BiologicalPhysics. Furthermore, class Momentum in GeneralPhysics would be different from Momentum in ClassicalMechanics.

In the future, many authors in the programming community may invent their own self-contained library systems like QuantumMechanics, QuantumPhysics, QuantumTheoryOfMatter for similar subjects in Physics. Then, it is highly desirable to set up a review board that will combine source codes into a single namespace like Science.Physics.QuantumMechanics. A library could best be built in such open source process. We believe that when such a processing scheme is established, a truly useful library system will evolve.

All classes of .NET framework are written in Basic, C++, and C#. It would be useful to build Science Code .Net in several languages, especially including Java. Although it is still questionable whether or not C# can be widely used in scientific computing, we use C# [16] for Science. We need not to be sensitive about which language is used, because it is not difficult to translate from one language to another.

To implement source codes in this way of Science Code .Net [17] is not easy, because library programmers should have more concrete knowledge about a subject. However, users who know concepts of a subject can write up a main program for a specific project quickly, because the same name of concepts are used in the library. Thus, the activity of coding with Science Code .Net can help students to enhance understanding concepts. Science Code .Net could provide us with maximum concept understanding with minimum time consumption. Although the road to reach the goal is difficult, still the library source codes presented here would be the first step to start. We expect that this library system is scalable and self-organizable.

Acknowledgement

The author thanks KIAS for associate membership.

References