**CodeChain Requirements**

You have just being persuaded by a charismatic friend to join a project that on the surface seems exciting and cutting edge. However, as you examine the specific requirement, the approach appears confusing, and in different ways, it makes no sense. The project is called CodeChain. A CodeChain is a list of blocks. The CodeChain can be modified in only one way, which is to append a block at the end of the chain. Blocks can be added only at the end of the list, and cannot be deleted. The core of each block in the chain is a piece of code. The code can be arbitrary and it can be invoked through an interface that intuitively looks like:

*result*1, *result*2, …, *resultn* = obj.foo(*arg*1, *arg*2, …, *argm*)

In other words, it consists of a method foo on an object obj whose arguments are *arg*1, *arg*2, …, *argm* and that returns *result*1, *result*2, …, *resultn*. The arguments can be results from the previous blocks. The requirements stress the point that the proposed interface is conceptual and it is not mandating a specific syntax. In fact, you are responsible for designing the syntax. Another requirement is that in addition to returning *result*1, *result*2, …, *resultn*, the function foo can have side effects, such as sending an email or appending other blocks to the CodeChain. In short, block execution boils down to the ability of executing the arbitrary code in the block with modalities that you will design. However, a block is never executed in isolation. The CodeChain supports only sequential execution of the chain, which means that each block in the list is executed in order.

Some desirable features of CodeChain should ideally be that:

1. The type of an object (such as *result*1 or *arg*1) should be known more precisely than just Object or a nonspecific class,

2. Reflection and downcasting should be avoided if possible, and

3. Functions can have arbitrary names, which should be more descriptive than foo or any other fixed convention.

However, it may be impossible to achieve all these three features. The project should attempt to achieve as many as possible while striving for good craftsmanship.

**Use Cases**

Upon hearing these requirements, you are aghast. There are sections that do not make much sense, but luckily, the requirements also come with a few use cases. It is important to point out that the uses cases are only some examples of the ways in which a CodeChain can be applied, do not exhaust all potential applications of code chains, and you are not required to design and implement any one of the use cases.

• The block code can take no arguments

• Given a student and a list of courses, the code automatically enrolls the student in as many courses as possible as long as she does not exceed eighteen credit hours.

• Upon verifying that a student has made the proper payment, send a confirmation email to the RA and adds a specific block at the end of the CodeChain.

• A function of one argument that returns only one result

Although these use cases may be useful to give a concrete idea of the requirements and for incorporating in testing, they are not part of your design objective. Rather, you are asked to design an extensible framework that would accommodate application as future developers think of additional uses for CodeChain.

Although the use cases partly clarify the requirements, as you have learned in Section 5.1, you fully expect the design process to be sloppy, non-deterministic, based on heuristics, and iterative (and possibly very frustrating). However, you also shoot for a product that is tidy, clearly addresses the relevant trade-offs, priorities, and restrictions: in other words, it will be a beautiful architecture.