Стажування
Software/C#

Create a plugin library. Plagin is a generic class, where generic`s type is a type, which this class can transform.
Each plugin contains method with parameter of plugin`s generic type and return modified value of the same type.

Example:

```
public string Modify(string param)
{
   return param.ToLower();
}
```

Implement few plugins for build-in types (eg: absolute value for numbers, convert to UTC from local for DateTime, etc).

Create class, that contains a collection of plugins and is a plugin. In this class, the modify method will apply all plugins from collection to input.

Create a class, that will be a base class for all classes, which will work with plugins. This class must have a generic parameter, which shows plugin type. Class must has a plugin property, data property and method for data output (eg. Print to console). Before output, data must be modified by plugin.

Create a class, that will be a plugin and a pluginable at the same time. In it`s Modify method it will modify data by itself and by plugin (eg. Multiply number by 2 and apply plugin).

Create a simple console demo.

Glossary

Generator - http://en.wikipedia.org/wiki/Generator_(computer_programming)
Solve these tasks using one of these platforms:
1. Silverlight
2. WPF
3. ~~WinRT Metro~~ Windows Store App

Task 1

Create method Clone(), which returns a copy of object

Examples:

```
4.Clone();
var str = "asdfasdf".Clone();
var v = new object().Clone();
```

Task 2

Create set of methods for Task and Task<T>, each of them will be similar to ContinueWith() from TPL, but will execute continuation in main (UI thread).

Overload method with parameters:
- Action
- Action<Task> where parameter is a task, for which continuation is created
- Func<T> / Func<Task<T>>
- Func<Task, T> / Func<Task, Task<T>> where parameter is a task, for which continuation is created

Method returns Task(for overloads with Action) or Task<T> (for overloads with Func, Result is a value, returned by continuation). Returned task must finish only after main Task and continuation are completed.

Task 3

Create mutex using tasks and async/await. The main difference with System.Threading.Mutex is that this mutex must not block current thread (for using in UI thread).

Mutex`s interface:
Lock() – returns a Task, wich will be completed only when mutex will become free. This task must be unique per lock to disallow more than 1 unblock per Release() call.
Release() – release mutex. One and only one of tasks, returned by Lock(), must become completed.

Examples:

```
await mutex.Lock();
// critical code
mutex.Release();
```

Task 4

To mutex from task 3 add method to allow using mutex by way, showed in example below.

```
using(await mutex.LockSection())
{
    //critical code
}
```

Task 5

Create generator of random number sequence. Generator must return a sequence of integer numbers from O to N, where N is a method`s parameter.

As a source of numbers use random.org API http://www.random.org/integers/?num=10&min=1&max=6&col=1&base=10&format=plain&rnd=new, where Max parameter is a max possible number, returned by generator.

Method must cache result into a local queue and return numbers from it. If queue becomes empty – make an async request to API to populate queue, and return numbers from BCL`s Random generator.

In case of API unavailability or error, stop making requests and return numbers from BCL`s generator only.

Task 6

For each assembly in AppDomain and for each assembly, that will be loaded in future, collect dictionary, where key is Type and value is reference to default ctor. If there is no default ctor, ignore type.

Create method for class Type called Create(), wich will return new object of that type using cached ctor.

Background assembly scanning in multiple threads is a plus.

Create must be thread-safe.