How do I design a class so that methods must be called in a certain order?

devblogs.microsoft.com/oldnewthing/20190318-00

March 18, 2019

Raymond Chen

Suppose you have a class with some methods, and the methods must be called in a certain order. For example, suppose your class is for importing photos from a camera. First, you have to specify which camera you are importing from. Then you discover the pictures. Then you have to select which pictures you want to import, and set the options for how you want the download to occur. And finally, you download them.

```
class PhotoImporter
{
public:
 PhotoImporter(const Camera& camera);

 // Optionally configure the discovery.
 void SetFindOnlyNewPhotos(bool newPhotosOnly);

 // Call this first to get some photos.
 std::vector<Photo> DiscoverPhotos();

 // Then decide which photos you want to download by calling
 // this for each photo you want to download.
 void SelectPhoto(Photo photo);
 void SelectAllDiscoveredPhotos();

 // Configure the download. You must set a download folder.
 // The other settings default to false.
 void SetDownloadFolder(const path& downloadFolder);
 void SetRenumberPhotos(bool renumber);
 void SetDeleteOriginalsAfterDownload(bool deleteOriginals);

 // And then download them.
 void Download();
};
```

The problem is that there is nothing preventing the caller from calling the methods out of order or omitting some methods altogether.

```
void confused()
{
 PhotoImporter importer(mainCamera);
 importer.SelectAllDiscoveredPhotos();
 importer.Download();
 importer.SetRenumberPhotos(true);
 importer.DiscoverPhotos();
 // never specified a download folder
}
```

One trick for making it harder to call the methods in the wrong order is to represent each state of the import process as a separate class.

```
class PhotoImporter
{
public:
 PhotoImporter(const Camera& camera);

 // Optionally configure the discovery.
 void SetFindOnlyNewPhotos(bool newPhotosOnly);

 // Call this first to get some photos.
 DiscoveredPhotos DiscoverPhotos();
};

class DiscoveredPhotos
{
public:
 // Not publically constructible.

 const std::vector<Photo>& GetPhotos();

 // Decide which photos you want to download by calling
 // this for each photo you want to download.
 void SelectPhoto(Photo photo);
 void SelectAllDiscoveredPhotos();

 // Configure the download. You must set a download folder.
 // The other settings default to false.
 void SetDownloadFolder(const path& downloadFolder);
 void SetRenumberPhotos(bool renumber);
 void SetDeleteOriginalsAfterDownload(bool deleteOriginals);

 // And then download them.
 void Download();
}
```

Breaking it up this way means that it is impossible to call `Download` before calling `DiscoverPhotos`, because in order to download the photos, you need to get a `DiscoveredPhotos` object, and the only way to get one of those is by calling `Discover-Photos`.

And then there's the issue of requiring a download folder. For that, we could make the mandatory portion an explicit parameter to the `Download` method. We used this trick when we required the `Camera` to be passed to the `PhotoImporter` constructor.

```
class DiscoveredPhotos
{
public:
 ...
 void Download(const path& downloadFolder);
}
```

And to ensure that people don't try to do things like call `SetRenumberPhotos` after `Download`, you could put all of the download options into an options class.

```
class DownloadOptions
{
public:
 void SetRenumberPhotos(bool renumber);
 void SetDeleteOriginalsAfterDownload(bool deleteOriginals);
};

class DiscoveredPhotos
{
public:
 // Not publically constructible.

 const std::vector<Photo>& GetPhotos();

 // Decide which photos you want to download by calling
 // this for each photo you want to download.
 void SelectPhoto(Photo photo);
 void SelectAllDiscoveredPhotos();

 // And then download them.
 void Download(const path& downloadFolder,
              const std::optional<DownloadOptions>& options = std::nullopt);
}
```

While this technique forces the programmer to satisfy prerequisites before calling a method, it doesn't prevent the programmer from trying to go backwards. For example, after calling `Download`, the programmer could go back and select some more photos and then call `Download` a second time.

If you want to disallow that, then I'm stumped. I can't think of something that prevents an object from being used after a particular method is called, with enforcement at compile time. Maybe you can think of something?

Raymond Chen

Follow