Project Thinking

Summary

Our model of a thread pool is simply consist of a task queue and the thread pool.The task queue contains tasks which wait for an available thread to be assigned into. When a thread gets available , then thread pool allows the task queue to insert the front task from its queue. If a task is done , it is erased from the task queue.

Covered Patterns

Following patterns are covered in this project :

1) Singleton Pattern:

There is one thread pool so we used singleton pattern while implementing ‘ThreadPool’ class. It helped to access the class methods without actually instantiating it.

2) Composite Pattern:

Tasks which are need to be done can consist of another sub tasks. Both MultiTask and Task classes use ‘assignToPool()’ method which is from AbstractTask class so this composite pattern uses ‘safe’ implementation . For example, ‘shut down the computer’ task includes tasks such as ‘shut all running programs’ or ‘display “Goodbye”’.Therefore , there is an abstract base class ‘AbstractTask’ for both ‘Task’ and ‘MultiTask’ classes.

<u>Participants :</u>

Component : AbstractTask

Composite : MultiTask

Leaf : Task

3) Comand Pattern :

Because tasks can not perform their action without a thread which needs a threadpool , in this model ThreadPool class invokes AbstractTask classes to do their action by using Thread class. The thread pool call ‘Execute()’ method in each thread it contains , then that method of threads calls ‘Action()’ method of each task. Therefore , the ThreadPool class acts as an invoker , Thread classes acts as commands and AbstractTask classes act as a receiver.

<u>Participants :</u>

Receiver : AbstracTask

Command : Thread

Invoker : ThreadPool

4) Iterator Pattern :

The model uses iterator pattern to travers Task and Thread objects. Task iterators use MultiTask class as a collection and Thread iterators use ThreadPool class as a collection.

5) Observer Pattern :

In this model the thread pool observes threads and updates itself according to upcoming notifications from threads. ‘Update’ method of the thread pool notifies the taskqueue if there is an available thread. A memory manager also observes the thread pool to keep track of total memory used by all threads. If a thread changes its memory usage then the thread pool notifies the memory manager about that change.

Participants :

Subject : Thread (for ThreadPool)

ThreadPool (for TaskQueue and MemoryManager)

Observer : ThreadPool (of Thread)

TaskQueue (of ThreadPool)

MemoryManager (of ThreadPool)

Classes

class Iterable :

Interface for iterable objects.Iterators can be used on any class derived from this.Just a symbolic class to achieve hierarchy.

class AbstractObserver : public Iterable

Abstract class for observer objects.Includes **Update(int)** method.

class AbstractTask : public Iterable

Abstract class for Task and MultiTask classes.They represent tasks in the model.

Tasks in the taskqueue can assign themselves into the threadpool directly if there is a notification from the threadpool.

class Thread : public Iterable

Thread class which represents threads in the model. They are two kinds of thread which are heavy ones(with priority 1) and light ones (with priority 5). Threads have their Task field which is executed by threads and perform their jobs. After a task is finished doing its job , thread makes itself empty and notifies the threadpool with code 1.Then if there is any proper task available in the job queue , assigns itself to the thread. Contains:

class AbstractIterator

Base class for iterator classes.Basic iterator methods are declared here.

class ThreadIterator : public AbstractIterator

Concrete Iterator class for traversing threads.ThreadIterator class uses the thread pool as a container of threads.

class ThreadPool : public AbstractObserver

This is the class which represents the threadpool in the model. It is responsible for creating , executing and observing threads.The threadpool creates spesific number of both kinds of threads.After the creation, it notifies (with code 1) the taskqueue so the queue assigns tasks into the threads if there are any proper ones. Then , the threadpool executes all threads continuosly. If there any thread completes its task, then the threadpool gets a notify and it notifies the taskqueue about the empty thread.

class Task : public AbstractTask

Concrete task class in the model and it represents leaf component of composite pattern.

Task instances can assign themselves into a thread from the threadpool according to their memory requirements. A task needs spesific amount of time period to be done. When the thread which the task is assigned execute itself , Action() method of the task is invoked. Therefore time period for the task to end is decreases. When total time needed for the task is passed , the task gets marked as “completed” and gets discarded by the thread.

class TaskIterator : public AbstractIterator

Concrete Iterator class for traversing task and multitask instances.TaskIterator class uses a multitask as a container of tasks.

class MultiTask : public AbstractTask

Concrete task class in the model and it represents composite component of composite pattern.

Multitask instances have assignToPool() which is the same signutre method with Task class. However , in multitask object it assigns inner tasks of it into threads. When every inner Task instances are assigned , then the MultiTask instance is marked as assigned. Additionally , Action () method of MultiTask instances does nothing.(Transparent imlementation)

class TaskQueue : public AbstractObserver

Job/Task queue in the model. Contains tasks they which wait to be assigned into the threadpool.

When the taskqueue gets a notification from the threadpool , it assigns the task which is head of the line. The queue implemented in FIFO approach. If the task which is head of the line does not get assigned , then the other tasks have to wait until the assignment. When a task is assigned , it is erased from the line. If assignment is completed , the taskqueue considered that task is finished.

class MemoryManager : public AbstractObserver

MemoryManager class keeps track of total memory used by the threadpool.When a MemoryManager instance gets a notification , it updates its thread container and calculates total memory used by all of them. If total memory exceeds 1GB of memory, then creates a new log and write into “log.txt” file with all tasks running currently.