CHEAT SHEET

This cheat sheet is designed as a way for you to quickly study the key points of this chapter.

Threads

- Create a thread object by sending to the constructor a delegate that will be the thread's main method.
- Start the thread by calling explicitly the start method on the thread object.
- To use the thread pool, call `ThreadPool.QueueUserWorkItem` or `ThreadPool.RegisterWaitForSingleObject`.

Tasks

- Create a task object by sending to the constructor a delegate that will be the tasks main method. Start the task by calling explicitly the start method on the task object.
- Call one of the `Task.Run` static methods and send as parameter a delegate that will be tasks main method. The task is started automatically.
- Call one of the `Task.Factory.StartNew` static methods and send as parameter a delegate that will be the task's main method. The task is started automatically.

Locks

- Use `Monitor.Enter` or `Monitor.TryEnter` to acquire a lock, and use `Monitor.Exit` to release the lock.
- Use the `lock` statement in C#.

Cancellations

- Use `CancellationTokenSource` objects to control cancelable operations.
- Use the `CancellationToken` obtained from a `CancellationTokenSource` object via the `Token` property to start a cancelable operation
- Call `Cancel` on the `CancellationTokenSource` object to initiate the cancellation.
- Inside the cancelable operation stop what you were doing to cancel the operation.

async/await

- These are two new keywords introduced in C# 5.0
- `async` marks a method as asynchronous, effectively telling the compiler that the method will have an await instruction in the body. If you don't have any `await` instruction in the method body, the compiler will issue a warning.
- `await` blocks the execution of the current method, waiting for the awaited operation to complete.

BackgroundWorker

- Wire the long-running method that you need executed via the `DoWork` event.
- Start the operation by calling `RunWorkerAsync`.
- To find out when the long operation is done, subscribe to the `RunWorkerCompleted` event.
- To get information about the progress of the operation, subscribe to the `ProgressChanged` event.

Task continuation

- Continue a task by invoking `ContinueWith`, `WhenAll`, or `WhenAny` methods.

REVIEW OF KEY TERMS

asynchrony Operations that are run in a nonblocking fashion. When a method needs to call another method that potentially can block, instead of calling that method directly you can apply different techniques to avoid the blocking of the calling method.

Asynchronous Pattern Model (APM) When using this pattern, a method is split in two parts, a `Begin` and an `End` part. The begin part is responsible to prepare the call and to return the caller right away, and the end part is the one called to get back the result. The method was run in a thread from the thread pool. It is not recommended to use this approach for new development; instead use the new TAP.

atomic operation An operation that will be run at once without being interrupted by the scheduler.

deadlock Occurs when two or more threads try to acquire a lock on a resource that one of the other threads has locked already; neither of them can make more progress. There are four conditions that need to be fulfilled and that lead to a deadlock: mutual exclusion, hold and wait, no preemption, and circular wait.

Event-based Asynchronous Pattern (EAP) This pattern requires a method to be suffixed with `Async` and provide events and delegates to signal when the method finished or failed. It is not recommended for new development to use this approach; instead use the new TAP.

fork-join pattern The process of spawning another thread from the current thread (fork) to do something else while the current threads continue their work and then to wait for the spawned thread to finish its execution (join).

multithreading The capability of an operating system, or a hardware platform to have several threads of execution at the same time.

mutual exclusion Mutual exclusion is the problem, first solved by Edsger W. Dijkstra, of ensuring that two threads can't be in the same critical section at the same time.

race condition Occurs when two or more threads access shared data, writing at the same time. If the access to data is for read purposes only, there is no problem. But when several threads try to write, one thread might overwrite the data written by another thread, not taking in consideration the change.

scheduler A component of the operating system that ensures that threads are given access to the CPU in a fair manner, avoiding situations when a thread monopolizes the CPU.

task A unit of work. It normally represents an asynchronous operation that is part of a bigger problem.

Task Parallel Library (TPL) A .NET library created by Microsoft that tries to abstract away and simplify the code that deals with threads.

Task-based Asynchronous Pattern (TAP) A pattern based on a single method that returns `Task` or `Task<Result>` objects that represent the asynchronous work in progress. This is the recommended pattern for the new development.

thread The smallest unit of execution that can be independently scheduled by the operating system.

thread pool The thread pool represents a pool of precreated threads that can be used by the tasks, or to queue work items, or to run asynchronous I/O operations.

EXAM TIPS AND TRICKS

You can print the Review of Key Terms and the Cheat Sheet for this chapter to help you study. You can find these files in the ZIP file for this chapter at `www.wrox.com/remtitle.cgi?isbn=1118612094` on the Download Code tab.