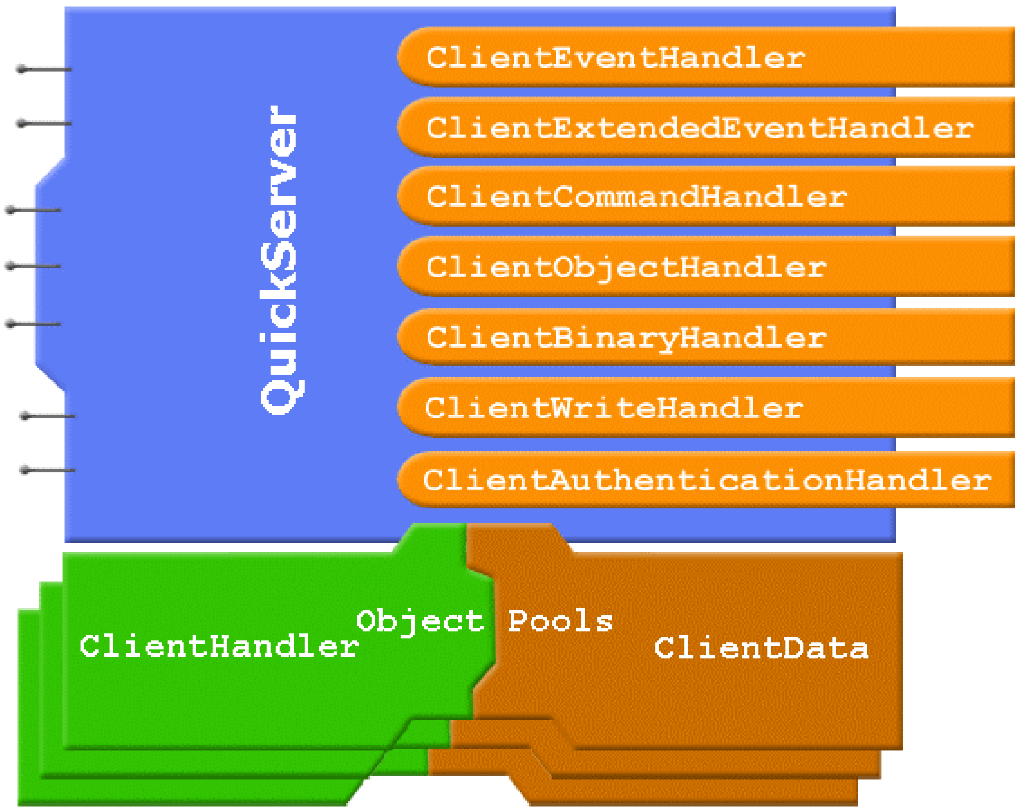
# QuickServer 1.4.6 - *Basic Architecture*

empowering protocols

Below diagram shows basic architecture of QuickServer framework. The seven spokes on the QuickServer block are the seven Service interface methods.



`QuickServerConfig` may also be directly used in `initService()` to configure QuickServer.

Note: QSAdminServer is not shown in this diagram, it is a composed QuickServer within the main QuickServer.

Of the eight components/class (application specific implementations) connected to QuickServer block only `one of the (#)` class is the absolutely necessary class (see below).

`ClientHandler` implementation object is used from the pool of objects for every client connected; optional `ClientData` class is associated with the `ClientHandler` class. Threads are picked from pool and used to execute any task that `ClientHandler needs to perform.`

`ClientHandler` object contain references to

- `ClientEventHandler` (optional)
- `ClientCommandHandler` (#)
- `ClientObjectHandler` (#)
- `ClientBinaryHandler` (#)
- `ClientWriteHandler` (optional)
- `ClientAuthenticationHandler` (optional)
- ClientExtendedEventHandler (optional)

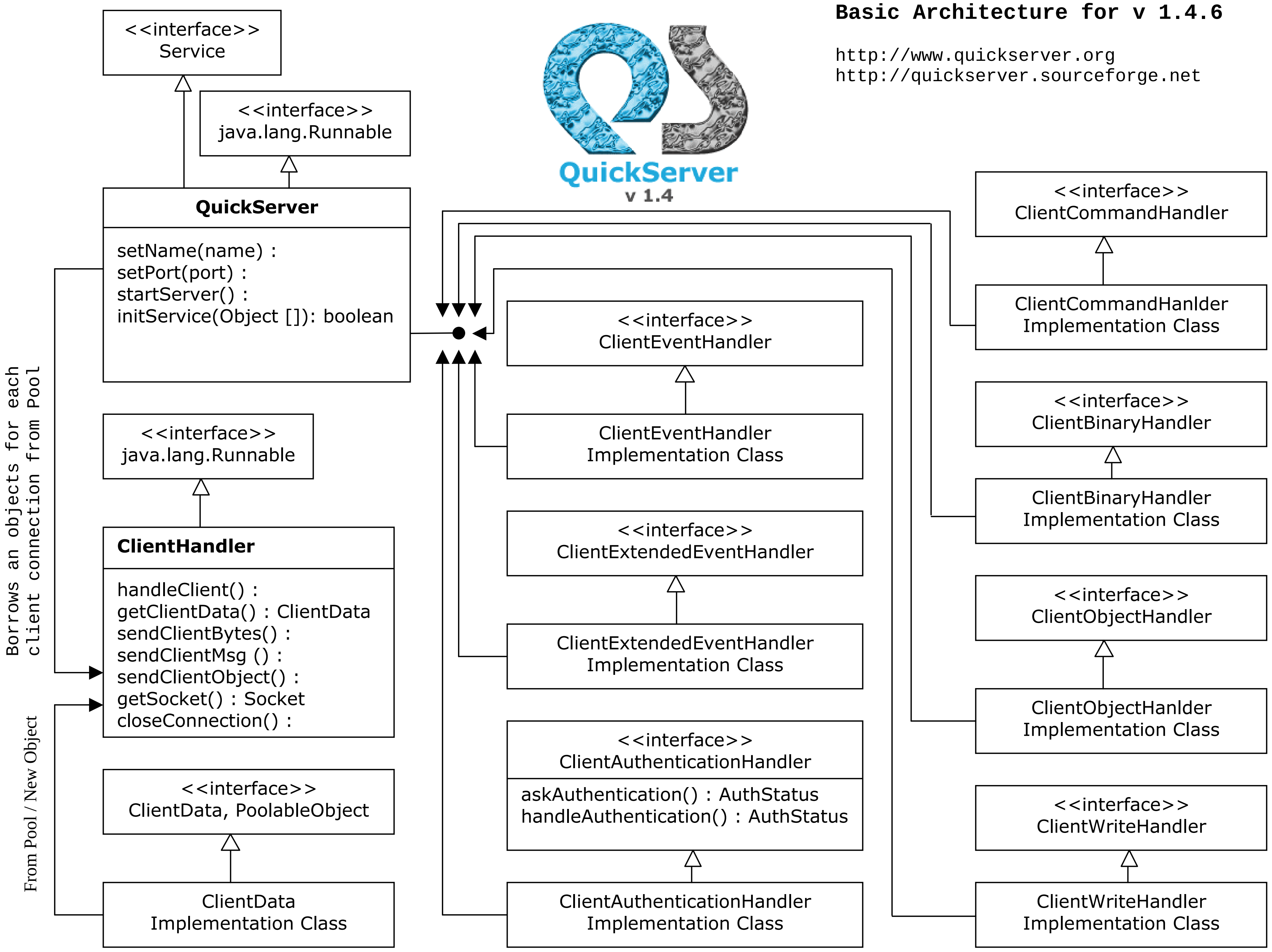
objects created by QuickServer when it starts.

[#] = Any one of these has to be set based on default DataMode for input.

`QuickServerConfig` object is constructed by `initService()` method of the Service interface, that QuickServer implements, after reading configuration from XML file.

# Basic Architecture for v 1.4.6

http://www.quickserver.org
http://quickserver.sourceforge.net



QuickServer v 1.4
<<interface>> Service
<<interface>> java.lang.Runnable
QuickServer
setName(name) :
setPort(port) :
startServer() :
initService(Object []): boolean
<<interface>> java.lang.Runnable
ClientHandler
handleClient() :
getClientData() : ClientData
sendClientBytes() :
sendClientMsg () :
sendClientObject() :
getSocket() : Socket
closeConnection() :
<<interface>> ClientData, PoolableObject
ClientData Implementation Class
Borrows an objects for each client connection from Pool
From Pool / New Object
<<interface>> ClientEventHandler
ClientEventHandler Implementation Class
<<interface>> ClientExtendedEventHandler
ClientExtendedEventHandler Implementation Class
<<interface>> ClientAuthenticationHandler
askAuthentication() : AuthStatus
handleAuthentication() : AuthStatus
ClientAuthenticationHandler Implementation Class
<<interface>> ClientCommandHandler
ClientCommandHanlder Implementation Class
<<interface>> ClientBinaryHandler
ClientBinaryHandler Implementation Class
<<interface>> ClientObjectHandler
ClientObjectHanlder Implementation Class
<<interface>> ClientWriteHandler
ClientWriteHandler Implementation Class

## QuickServer - *Threading Model*

For every server instance there will be

- Only one instance of QuickServer
- Only one instance of GhostSocketReaper (if timeout is > 0)
- Only one instance of any/all business classes like ClientEventHandler, ClientCommandHandler, ClientObjectHandler, ClientBinaryHandler, ClientWriteHandler, ClientAuthenticationHandler, ClientExtendedEventHandler
- Every client connected will have a ClientHandler implementation associated with it based on the server mode
    - Blocking Mode = BlockingClientHandler
    - Non Blocking Mode = NonBlockingClientHandler
- If ClientData is set, then every ClientHandler will have an instance of ClientData associated with it. The ClientData objects will be polled if it implements PoolableObject interface.
- Based on the server mode, threads will be associated with ClientHandler to execute any processing for the client i.e.;
    - In Blocking Mode: A thread is dedicated to every ClientHandler for processing events from the client.
    - In Non Blocking Mode: A thread is associated with a ClientHandler only as needed or when data is available for processing.
        - When ever any data is available for reading the thread is assigned with READ as ClientEvent.
        - If client registers for any write event, then whenever the data can be written a thread is assigned with WRITE event for processing any writes.

In the same JVM, there can be more than one instance of QuickServer object that is running based on the implementation. A typical QuickServer setup will have one QuickServer object for the main server and another for the QSAdminServer.