JextLoader acts as a local IPC service to launch or terminate Jext editor instances within one JVM.
It generates a randomized port and authorization key, writes them to a file, then listens on a ServerSocket.
On each connection, it verifies the client's origin and message before invoking Jext APIs.

- **Port & Key Generation**
  - Selects a random port in the range 0–16382.
  - Generates a 30-bit random authorization key.
  - Persists both values into an `.auth-key` file.
- **Connection Loop**
  - Continuously accepts incoming socket connections.
  - Validates client IP (`127.0.0.1`) to enforce localhost-only access.
  - Reads a single-line command, then closes the reader immediately.
- **Command Parsing**
  - Recognizes two message formats:
    - `load_jext:<args>?:…:<key>` to open files or windows.
    - `kill:<key>` to request application shutdown.
  - Uses `StringTokenizer` on the substring between prefix and key to extract load arguments.
- **Instance Dispatch**
  - If `jextLoader.newWindow` is true, always launches a fresh window.
  - Otherwise, reuses the first open `JextFrame` to open files.
  - On kill requests, checks background mode and visible windows before cleanup and exit.

- **Fields**
  - `int port` & `String key`: govern socket address and authorization.
  - `File auth`: points to the `.auth-key` file on disk.
  - `ServerSocket server`: listens for local commands.
  - `Thread tServer`: drives the asynchronous accept loop.
- **State Transitions**
  i. **Initialization**: constructor writes auth file, opens ServerSocket, starts `tServer`.
  ii. **Serving**: `run()` loops until `tServer` is null, handling valid or intruding connections.
  iii. **Shutdown**: `stop()` interrupts the thread, closes the socket, deletes the auth file.

Method	Purpose	Key Parameters	Return Value
`JextLoader()`	Generate auth key/file, bind ServerSocket, start thread.	none	— (constructor)
`stop()`	Halt listener thread, close socket, delete auth file.	none	`void`
`run()`	Main loop: accept connections, validate, dispatch actions.	none	`void`

- **Swallowed Exceptions**: Many `catch(IOException)` blocks lack logging or remediation.
- **Hardcoded Boundaries**: Uses `127.0.0.1` and fixed port offset without configuration.

- **Thread Safety**: Volatile mutation of `tServer` without synchronization.
- **Deprecated Patterns**: Reliance on `Vector` and `StringTokenizer` in modern Java.
- **UI in Service**: Invoking `JOptionPane` in headless environments risks deadlocks.
- **Security Exposure**: Auth file stored with no file-permission checks; key in cleartext.