The core difference between **Java Class Loaders** and **Windows EXE Loaders** lies in **what** they load and **when** they perform the loading, reflecting the fundamental difference between Java's virtual machine architecture and Windows' native operating system architecture.

| Feature | Java Class Loader | Windows EXE Loader (PE Loader) |
| --- | --- | --- |
| **Target Code** | **Bytecode** (.class files or JARs) | **Native Machine Code** (PE file format: .exe, .dll) |
| **Execution Context** | **Java Virtual Machine (JVM)** | **Operating System Kernel** (Creates a new OS process) |
| **Load Timing** | **Dynamic (On-Demand)**. Classes are loaded and linked only when they are first referenced by the running application. | **Static (Initial)**. The entire executable and its statically linked/imported DLLs are mapped into memory *before* the program starts executing its main entry point. |
| **Class Isolation** | High. Uses a **Hierarchical Delegation Model** to manage namespaces, allowing multiple versions of the same class to exist in different ClassLoaders (e.g., in web servers). | Low. Loads modules into a single process's address space. DLL resolution is global within that process. |
| **Relocation/Linking** | **Linking** (Verification, Preparation, Resolution) happens at runtime, inside the JVM, on the bytecode. | **Relocation** and **import resolution** (DLL finding) happen at the initial load time by the OS loader. |