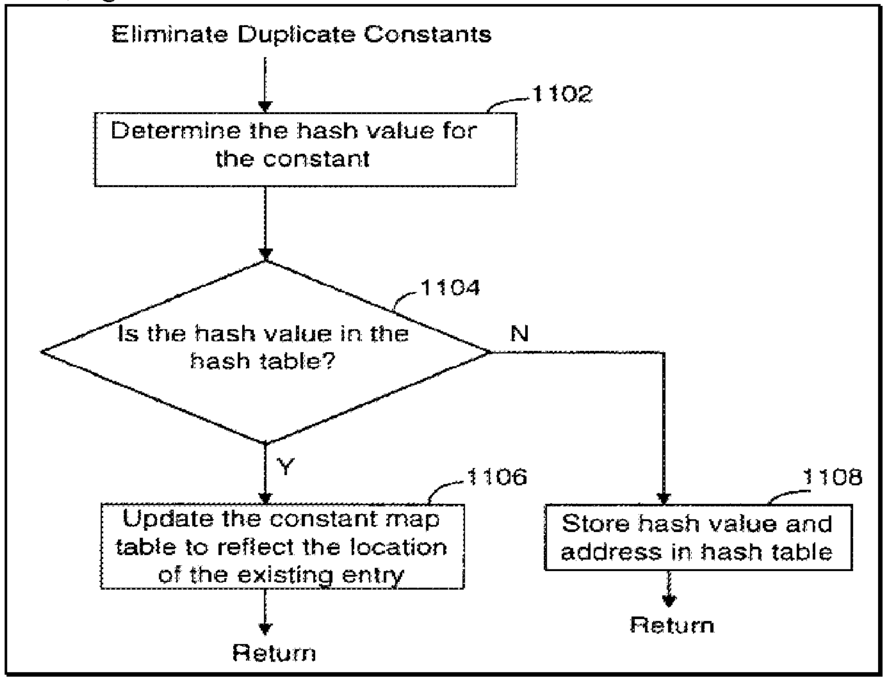
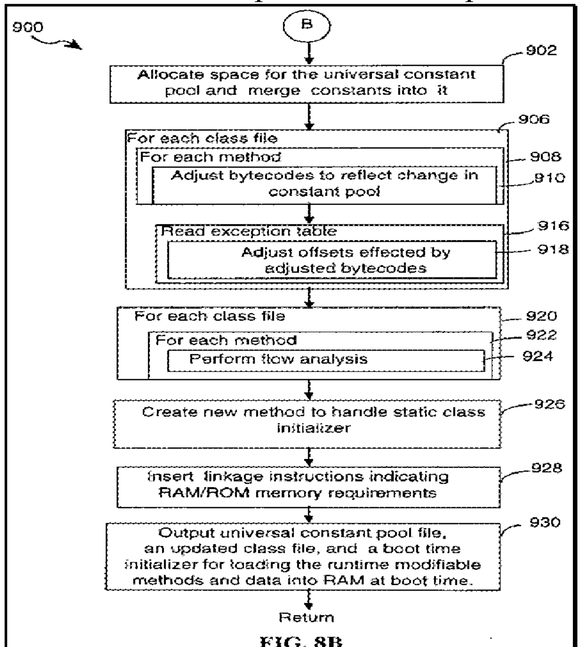
Exhibit 4 - Request For Reexamination of U.S. Patent No. 5,966,702

U.S. Patent No. 5,815,718
Inventor: Theron D. Tock
Issue Date: Sep. 29, 1998
Filing Date: May 30, 1996
(“*Tock*”)

U.S. Patent No. 5,966,702 – Claim 1	***Tock***
1. A method of pre-processing class files comprising:	*Tock* discloses a method of pre-processing class files. The class loader described in *Tock* performs pre-processing on class files as described below. “In summary, this disclosure pertains to an offline ***class loader*** that is used to produce an executable module whose ***classes*** are preloaded into memory without requiring runtime dynamic loading. The executable module, nevertheless, contains a class structure that is tailored for runtime dynamic loading. Thus, the offline class loader modifies the existing class structures to accommodate static loading.” *Tock*, col. 1, ll.41–47 (emphasis added).

determining plurality of duplicated elements in a plurality of class files;	*Tock* discloses determining a plurality of duplicated elements in a plurality of class files. As illustrated and described below, *Tock* discloses scanning each entry in the class's constant pool to identify duplicates. *Tock*, fig. 10.  Eliminate Duplicate Constants 1102 Determine the hash value for the constant 1104 Is the hash value in the hash table? N Y 1106 Update the constant map table to reflect the location of the existing entry Return 1108 Store hash value and address in hash table Return "Next, the offline class loader proceeds to ***eliminate duplicate constants***. This is performed in order to combine the constant pools of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is ***scanned for duplicate constants*** (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added).

forming a shared table comprising said plurality of duplicated elements;	*Tock* discloses forming a shared table comprising said plurality of duplicated elements. *Tock* uses a hash table to detect and remove duplicates from the combined constant pool. "Referring to FIG. 10, duplicate constants are detected by using a hash table. The hash value of the constant is determined by an appropriate hashing function (step 1102). A check is made to determine whether the hash value is contained in the hash table (step 1104). If the hash value exists in the hash table, then the constant is a duplicate and the entry is deleted from the ***constant pool*** by altering the constant's entry in the map table to reflect the memory location of the existing constant (step 1106). Otherwise, the constant's hash value and memory location are stored in the hash table (step 1108)." *Tock*, col. 8, ll. 45-55 (emphasis added).
removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and	*Tock* discloses removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files. *Tock* produces updated class files that are more compact because duplicates have been removed from the combined constant pool. "The output of the offline class loader 302 can consist of two files: a constant pool file containing the constant data for the entire application; and an ***updated class file*** containing the class data structures and class members. The data in both of these files is formatted as data definitions, where each definition specifies a bytecode and an offset indicating a memory location. The updated class file will include the memory storage indicators which will indicate in which type of memory storage device a particular set of bytecodes is to reside. However, the method and system described herein is not limited to producing these two files. Other file configuration can be used including, but not limited to, a single file containing all the related class data." *Tock*, col. 5, ll.38–50 (emphasis added). "The offline class loader also performs a number of ***optimizations in order to produce a more compact representation*** of the executable code. For example, the ***constant pool*** that is associated with each class is ***combined*** for all the classes residing in the application." *Tock*, col. 5, ll.29–34 (emphasis added).

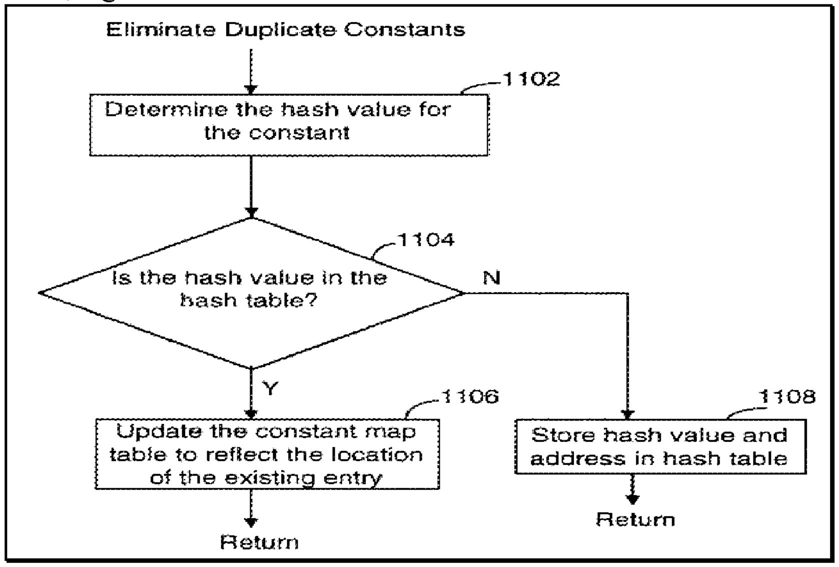
forming a multi-class file comprising said plurality of reduced class files and said shared table.	*Tock* discloses a multi-class file comprising said plurality of reduced class files. *Tock's* class loader can output the constant pool file and updated class file as a single file.  900 B Allocate space for the universal constant pool and merge constants into it 902 For each class file 906 For each method 908 Adjust bytecodes to reflect change in constant pool 910 Read exception table 916 Adjust offsets effected by adjusted bytecodes 918 For each class file 920 For each method 922 Perform flow analysis 924 Create new method to handle static class initializer 926 Insert linkage instructions indicating RAM/ROM memory requirements 928 Output universal constant pool file, an updated class file, and a boot time initializer for loading the runtime modifiable methods and data into RAM at boot time. 930 Return FIG. 8B *Tock*, fig. 8B. "Lastly, the offline class loader ***outputs*** the ***universal constant pool***, an ***updated class file*** containing the class data structures and the indicators specifying the memory storage requirements, as well as a special boot time indicator (step 930)." *Tock*, col. 10, ll.29–32 (emphasis added). "The output of the offline class loader 302 can consist of two files: ***a constant pool file*** containing the constant data for the entire application; and ***an updated class file*** containing the class data structures and class members. The data in both of these files is formatted as data definitions, where each definition specifies a bytecode and an offset indicating a memory location. The updated class file will include the memory storage indicators which will indicate in which type of memory storage device a particular set of bytecodes is to reside. However, the method and system described herein is not limited to producing these two files. Other file configuration can be used including, but not limited to, ***a single file*** containing all the related class data." *Tock*, col. 5, ll.38–50 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 5	
5. The method of claim 1, wherein said step of determining a plurality of duplicated elements comprises:	
determining one or more constants shared between two or more class files.	*Tock* discloses determining one or more constants shared between two or more class files. *Tock* describes scanning each class's constant pool for duplicate constants to be eliminated and merging the constants into a universal constant pool. "Next, the offline class loader proceeds to eliminate duplicate constants. This is performed in order to combine the constant pools of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is ***scanned for duplicate constants*** (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added). "Once space is allocated for the universal constant pool, each entry from the various class constant pools is ***merged*** into the ***universal constant pool*** (step 902)." *Tock*, col. 9, ll.27–29 (emphasis added).

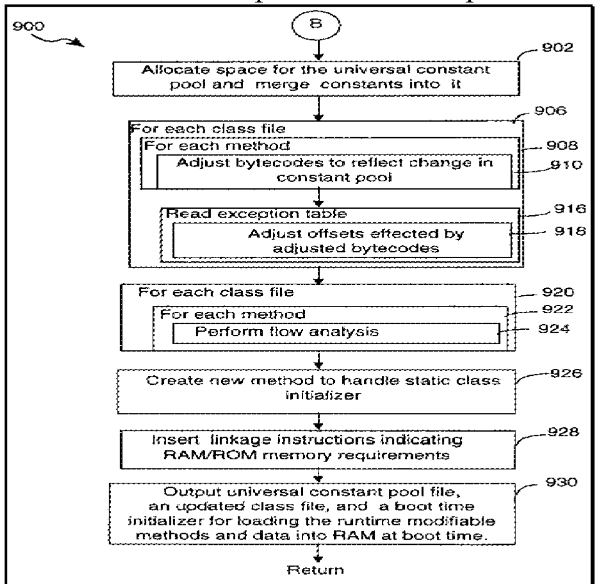
U.S. Patent No. 5,966,702 – Claim 6	
6. The method of claim 5, wherein said step of forming a shared table comprises:	
forming a shared constant table comprising said one or more constants shared between said two or more class files.	*Tock* discloses forming a shared constant table comprising said one or more constants shared between said two or more class files. The shared constant table is formed by merging the various class constant pools into a universal constant pool. "Once space is allocated for the universal constant pool, each entry from the various class constant pools is ***merged*** into the ***universal constant pool*** (step 902)." *Tock*, col. 9, ll.27–29 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 7	
7. A computer program product comprising:	*Tock* discloses a computer program product. *Tock's* description of a system and method for preloading a subset of classes includes a computer program product. "The present invention relates generally to object-oriented computer systems having classes that are dynamically loaded at runtime, and particularly to a system and method for preloading a subset of the classes in a read-only memory." *Tock*, col. 1, ll.4–7.

a computer usable medium having computer readable program code embodied therein for pre-processing class files, said computer program product comprising:	*Tock* discloses a computer usable medium having computer readable program code embodied therein for pre-processing class files. The class loader described in *Tock* performs pre-processing on class files as described below. "In summary, this disclosure pertains to an offline ***class loader*** that is used to produce an executable module whose ***classes*** are preloaded into memory without requiring runtime dynamic loading. The executable module, nevertheless, contains a class structure that is tailored for runtime dynamic loading. Thus, the offline class loader modifies the existing class structures to accommodate static loading." *Tock*, col. 1, ll.41–47 (emphasis added). "Referring to FIG. 1, a server computer typically includes one or more processors 112, a communications interface 116, a user interface 114, and memory 110. Memory 110 stores: • an operating system 118; • an Internet communications manager program or other type of network access procedures 120; • a compiler 122 for translating source code written in the Java programming language into a stream of bytecodes; • a source code repository 124 including one or more source code files 126 containing Java source code; • a class file repository 128 including one or more class files 130, and one or more class libraries 131 containing class files, each class file containing the data representing a particular class; • an offline class loader 132 which is used to preload a certain set of classes; the offline class loader can also be referred to as a static class loader; • an assembler 134 which produces an object file representing the class members, class data structures, and memory storage indicators in a format that is recognizable for the linker; • a linker 136 for determining the memory layout for a set of preloaded classes and for resolving all symbolic references; • a browser 138 for use in accessing HTML documents; and • one or more data files 146 for use by the server." *Tock*, col. 4, ll.6–34 (emphasis added).

computer readable program code configured to cause a computer to determine a plurality of duplicated elements in a plurality of class files;	*Tock* discloses computer readable program code configured to cause a computer to determine a plurality of duplicated elements in a plurality of class files. As illustrated and described below, *Tock* discloses scanning each entry in the class's constant pool to identify duplicates. *Tock*, fig. 10.  Eliminate Duplicate Constants 1102 Determine the hash value for the constant 1104 Is the hash value in the hash table? N Y 1106 Update the constant map table to reflect the location of the existing entry 1108 Store hash value and address in hash table Return Return "Next, the offline class loader proceeds to ***eliminate duplicate constants***. This is performed in order to combine the constant pools of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is ***scanned for duplicate constants*** (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added).

computer readable program code configured to cause a computer to form a shared table comprising said plurality of duplicated elements;	*Tock* discloses computer readable program code configured to cause a computer to form a shared table comprising said plurality of duplicated elements. *Tock* uses a hash table to detect and remove duplicates from the combined constant pool. “Referring to FIG. 10, duplicate constants are detected by using a hash table. The hash value of the constant is determined by an appropriate hashing function (step 1102). A check is made to determine whether the hash value is contained in the hash table (step 1104). If the hash value exists in the hash table, then the constant is a duplicate and the entry is deleted from the ***constant pool*** by altering the constant's entry in the map table to reflect the memory location of the existing constant (step 1106). Otherwise, the constant's hash value and memory location are stored in the hash table (step 1108).” *Tock*, col. 8, ll. 45-55 (emphasis added).
computer readable program code configured to cause a computer to remove said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and	*Tock* discloses computer readable program code configured to cause a computer to remove said duplicated elements from said plurality of class files to obtain a plurality of reduced class files. *Tock* produces updated class files that are more compact because duplicates have been removed from the combined constant pool. “The output of the offline class loader 302 can consist of two files: a constant pool file containing the constant data for the entire application; and ***an updated class file*** containing the class data structures and class members. The data in both of these files is formatted as data definitions, where each definition specifies a bytecode and an offset indicating a memory location. The updated class file will include the memory storage indicators which will indicate in which type of memory storage device a particular set of bytecodes is to reside. However, the method and system described herein is not limited to producing these two files. Other file configuration can be used including, but not limited to, a single file containing all the related class data.” *Tock*, col. 5, ll.38–50 (emphasis added). “The offline class loader also performs a number of ***optimizations in order to produce a more compact representation*** of the executable code. For example, ***the constant pool*** that is associated with each class is ***combined*** for all the classes residing in the application.” *Tock*, col. 5, ll.29–34 (emphasis added).

computer readable program code configured to cause a computer to form a multi-class file comprising said plurality of reduced class files and said shared table.	*Tock* discloses computer readable program code configured to cause a computer to form a multi-class file comprising said plurality of reduced class files and said shared table. *Tock's* class loader can output the constant pool file and updated class file as a single file.  900 B 902 Allocate space for the universal constant pool and merge constants into it 906 For each class file For each method 908 Adjust bytecodes to reflect change in constant pool 910 Read exception table 916 Adjust offsets effected by adjusted bytecodes 918 For each class file 920 For each method 922 Perform flow analysis 924 Create new method to handle static class initializer 926 Insert linkage instructions indicating RAM/ROM memory requirements 928 Output universal constant pool file, an updated class file, and a boot time initializer for loading the runtime modifiable methods and data into RAM at boot time. 930 Return FIG. 8B *Tock*, fig. 8B. "Lastly, the offline class loader ***outputs*** the ***universal constant pool***, ***an updated class file*** containing the class data structures and the indicators specifying the memory storage requirements, as well as a special boot time indicator (step 930)." *Tock*, col. 10, ll.29–32 (emphasis added). "The output of the offline class loader 302 can consist of two files: ***a constant pool file*** containing the constant data for the entire application; and ***an updated class file*** containing the class data structures and class members. The data in both of these files is formatted as data definitions, where each definition specifies a bytecode and an offset indicating a memory location. The updated class file will include the memory storage indicators which will indicate in which type of memory storage device a particular set of bytecodes is to reside. However, the method and system described herein is not limited to producing these two files. Other file configuration can be used including, but not limited to, ***a single file containing all the related class data***." *Tock*, col. 5, ll.38–50 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 11	
11. The computer program product of claim 7, wherein said computer readable program code configured to cause a computer to determine said plurality of duplicated elements comprises:	
computer readable program code configured to cause a computer to determine one or more constants shared between two or more class files.	*Tock* discloses computer readable program code configured to cause a computer to determine one or more constants shared between two or more class files. *Tock* describes scanning each class's constant pool for duplicate constants and merging constants into a universal constant pool. "Next, the offline class loader proceeds to eliminate duplicate constants. This is performed in order to combine the constant pools of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is ***scanned for duplicate constants*** (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added). "Once space is allocated for the universal constant pool, each entry from the various class constant pools is ***merged*** into the ***universal constant pool*** (step 902)." *Tock*, col. 9, ll.27–29.

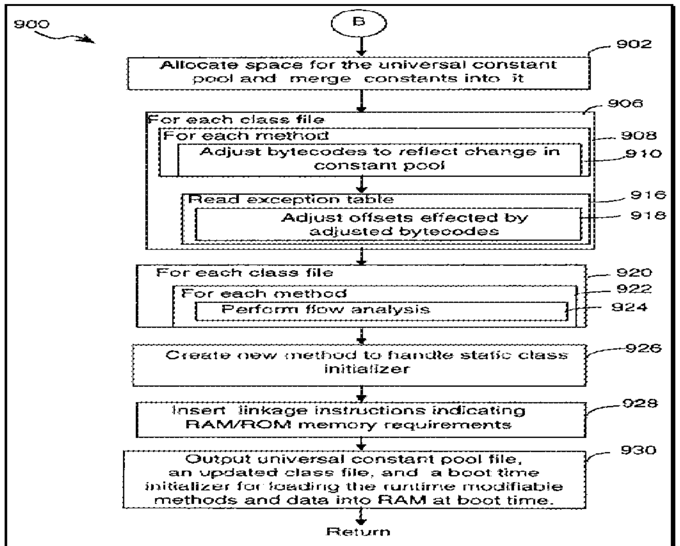
U.S. Patent No. 5,966,702 – Claim 12	
12. The computer program product of claim 11, wherein said computer readable program code configured to cause a computer to form said shared table comprises:	

computer readable program code configured to cause a computer to form a shared constant table comprising said one or more constants shared between said two or more class files.	*Tock* discloses computer readable program code configured to cause a computer to form a shared constant table comprising said one or more constants shared between said two or more class files. The shared constant table is formed by merging the various class constant pools into a universal constant pool. "Once space is allocated for the universal constant pool, each entry from the various class constant pools is ***merged*** into the ***universal constant pool*** (step 902)." *Tock*, col. 9, ll.27–29 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 13	
13. An apparatus comprising:	*Tock* discloses an apparatus. *Tock's* description of a server computer as shown below includes an apparatus. "Referring to FIG. 1, a server computer typically includes one or more processors 112, a communications interface 116, a user interface 114, and memory 110. Memory 110 stores: • an operating system 118; • an Internet communications manager program or other type of network access procedures 120; • a compiler 122 for translating source code written in the Java programming language into a stream of bytecodes; • a source code repository 124 including one or more source code files 126 containing Java source code; • a class file repository 128 including one or more class files 130, and one or more class libraries 131 containing class files, each class file containing the data representing a particular class; • an offline class loader 132 which is used to preload a certain set of classes; the offline class loader can also be referred to as a static class loader; • an assembler 134 which produces an object file representing the class members, class data structures, and memory storage indicators in a format that is recognizable for the linker; • a linker 136 for determining the memory layout for a set of preloaded classes and for resolving all symbolic references; • a browser 138 for use in accessing HTML documents; and • one or more data files 146 for use by the server." *Tock*, col. 4, ll.6–34.
a processor;	*Tock* discloses a processor. "Referring to FIG. 1, a server computer typically includes one or more ***processors 112***, a communications interface 116, a user interface 114, and memory 110. *Tock*, col. 4, ll.6–9 (emphasis added).

a memory coupled to said processor;	*Tock* discloses a memory. “Referring to FIG. 1, a server computer typically includes one or more processors 112, a communications interface 116, a user interface 114, and ***memory 110***. *Tock*, col. 4, ll.6–9 (emphasis added).
a plurality of class files stored in said memory;	*Tock* discloses a plurality of class files stored in said memory. “a class file repository 128 including ***one or more class files 130***, and one or more class libraries 131 containing class files, each class file containing the data representing a particular class” *Tock*, col. 4, ll.18–20 (emphasis added).
a process executing on said processor, said process configured to form a multi-class file comprising:	*Tock* discloses a process executing on said processor to form a multi-class file. *Tock’s* class loader processes class files. “an offline class loader 132 which is used to ***preload a certain set of classes***; the offline class loader can also be referred to as a static class loader; an assembler 134 which produces an object file representing the ***class members, class data structures, and memory storage indicators*** in a format that is recognizable for the linker; a linker 136 for determining the memory layout for a set of preloaded classes and for resolving all symbolic references” *Tock*, col. 4, ll.22–32 (emphasis added).

a plurality of reduced class files obtained from said plurality of class files by removing one or more elements that are duplicated between two or more of said plurality of class files; and	*Tock* discloses removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files. *Tock* produces updated class files that are more compact because duplicates have been removed from the combined constant pool. "The output of the offline class loader 302 can consist of two files: a constant pool file containing the constant data for the entire application; and an ***updated class file*** containing the class data structures and class members. The data in both of these files is formatted as data definitions, where each definition specifies a bytecode and an offset indicating a memory location. The updated class file will include the memory storage indicators which will indicate in which type of memory storage device a particular set of bytecodes is to reside. However, the method and system described herein is not limited to producing these two files. Other file configuration can be used including, but not limited to, a single file containing all the related class data." *Tock*, col. 5, ll.38–50 (emphasis added). "The offline class loader also performs a number of ***optimizations in order to produce a more compact representation*** of the executable code. For example, ***the constant pool*** that is associated with each class is ***combined*** for all the classes residing in the application." *Tock*, col. 5, ll.29–34 (emphasis added).

a shared table comprising said duplicated elements.	*Tock* discloses producing a shared table comprising duplicated elements. *Tock* describes removing duplicate constants in order to combine the constant pools of the classes.  900 B Allocate space for the universal constant pool and merge constants into it 902 For each class file 906 For each method 908 Adjust bytecodes to reflect change in constant pool 910 Read exception table 916 Adjust offsets effected by adjusted bytecodes 918 For each class file 920 For each method 922 Perform flow analysis 924 Create new method to handle static class initializer 926 Insert linkage instructions indicating RAM/ROM memory requirements 928 Output universal constant pool file, an updated class file, and a boot time initializer for loading the runtime modifiable methods and data into RAM at boot time. 930 Return FIG. 8B *Tock*, fig. 8B. "Next, the offline class loader proceeds to ***eliminate duplicate constants***. This is performed in order ***to combine the constant pools*** of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is scanned for duplicate constants (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added). "Lastly, the offline class loader outputs the ***universal constant pool***, an updated class file containing the class data structures and the indicators specifying the memory storage requirements, as well as a special boot time indicator (step 930)." *Tock*, col. 10, ll.29–32 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 15	
15. The apparatus of claim 13, wherein said duplicated elements comprise elements of constant pools of respective class files, said shared table comprising a shared constant pool.	*Tock* discloses that the duplicated elements comprise elements of constant pools of respective class files, said shared table comprising a shared constant pool. *Tock* describes eliminating duplicate constants and combining constant pools into a universal constant pool. "Next, the offline class loader proceeds to ***eliminate duplicate constants***. This is performed in order to combine the constant pools of all the classes in a space efficient manner. For each class file (step 806), each entry in the class' constant pool is ***scanned for duplicate constants*** (step 812). Referring to FIG. 10, duplicate constants are detected by using a hash table." *Tock*, col. 8, ll.40–46 (emphasis added). "Lastly, the offline class loader outputs ***the universal constant pool***, an updated class file containing the class data structures and the indicators specifying the memory storage requirements, as well as a special boot time indicator (step 930)." *Tock*, col. 10, ll.29–32 (emphasis added).

U.S. Patent No. 5,966,702 – Claim 16	
16. The apparatus of claim 13, further comprising:	*Tock* discloses a virtual machine having a class loader and a runtime data area, said class loader configured to obtain and load said multi-class file into said runtime data area. *Tock* describes a boot time initiator that loads a multi-class file into the runtime data area.
a virtual machine having a class loader and a runtime data area, said class loader configured to obtain and load said multi-class file into said runtime data area.	"Lastly, the offline ***class loader*** outputs the ***universal constant pool***, an ***updated class file*** containing the class data structures and the indicators specifying the memory storage requirements, as well a special ***boot time initiator*** (step 930). Referring to FIG. 12, the preloadable executable module and boot time initiator 1220 are permanently stored in the read-only memory of a client computer. Each time the client computer is powered on or rebooted, the boot time initiator 1220 is automatically executed. Among other tasks, the boot time initiator copies all methods and data that must be resident in random access memory during execution to the random access memory locations assigned to them by the linker." *Tock*, col. 10, ll.29–41.