US 20060026200A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2006/0026200 A1

Cabillic et al. (43) Pub. Date: Feb. 2, 2006

(54) METHOD AND SYSTEM FOR SHARED OBJECT DATA MEMBER ZONES

(75) Inventors: **Gilbert Cabillic**, Brece (FR); **Jean-Philippe Lesot**, Etrelles (FR)

Correspondence Address:
**TEXAS INSTRUMENTS INCORPORATED**
**P O BOX 655474, M/S 3999**
**DALLAS, TX 75265**

(73) Assignee: **Texas Instruments Incorporated**, Dallas, TX (US)

(21) Appl. No.: **11/187,199**

(22) Filed: **Jul. 22, 2005**

(30) **Foreign Application Priority Data**

Jul. 27, 2004 (EP) ..... 04291918.3

**Publication Classification**

(51) **Int. Cl.**
*G06F 17/00* (2006.01)

(52) **U.S. Cl.** ..... **707/103 R**

(57) **ABSTRACT**

Methods, computer-readable media, and systems for sharing duplicate data between objects in object-oriented applications are provided. In some illustrative embodiments, a method for sharing data member zones of objects in a software application executing on a processor is provided. The method includes instantiating a first object comprising a first plurality of data members and instantiating a second object comprising a second plurality of data members. The method further includes defining a first shared data member zone comprising a first portion of the first plurality of data members and a second portion of the second plurality of data members, and modifying a value of a data member in the first portion, the modification making the value available to a read access of a corresponding data member in the second portion.

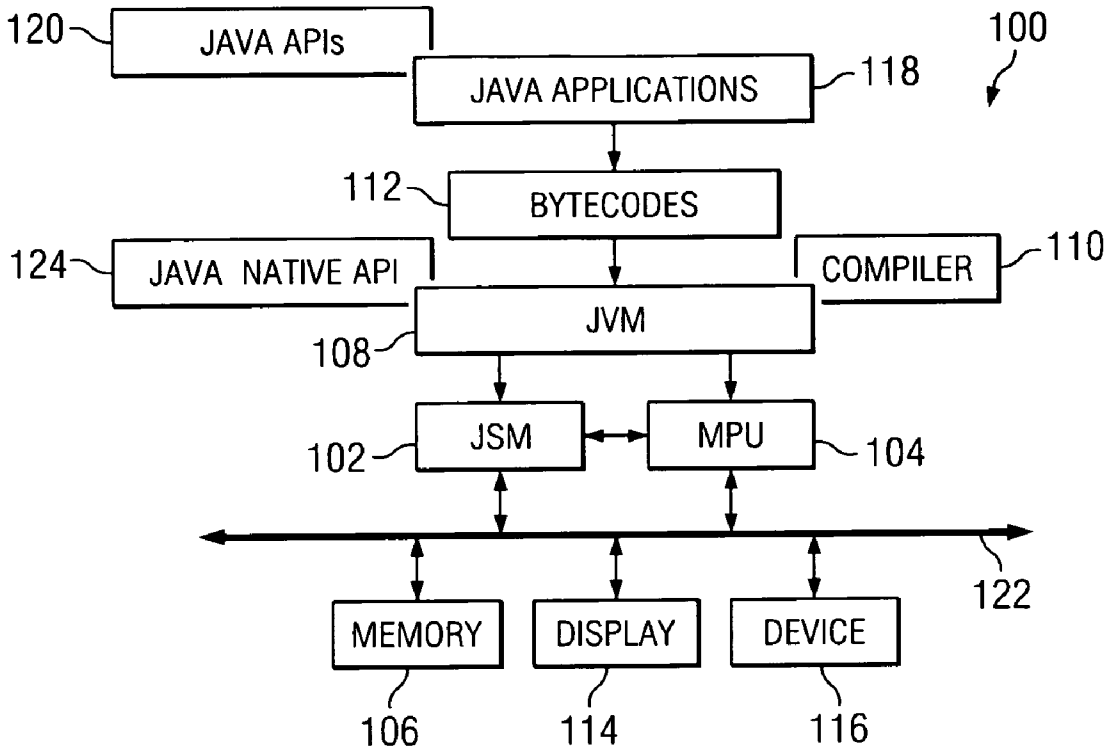


*FIG. 1*

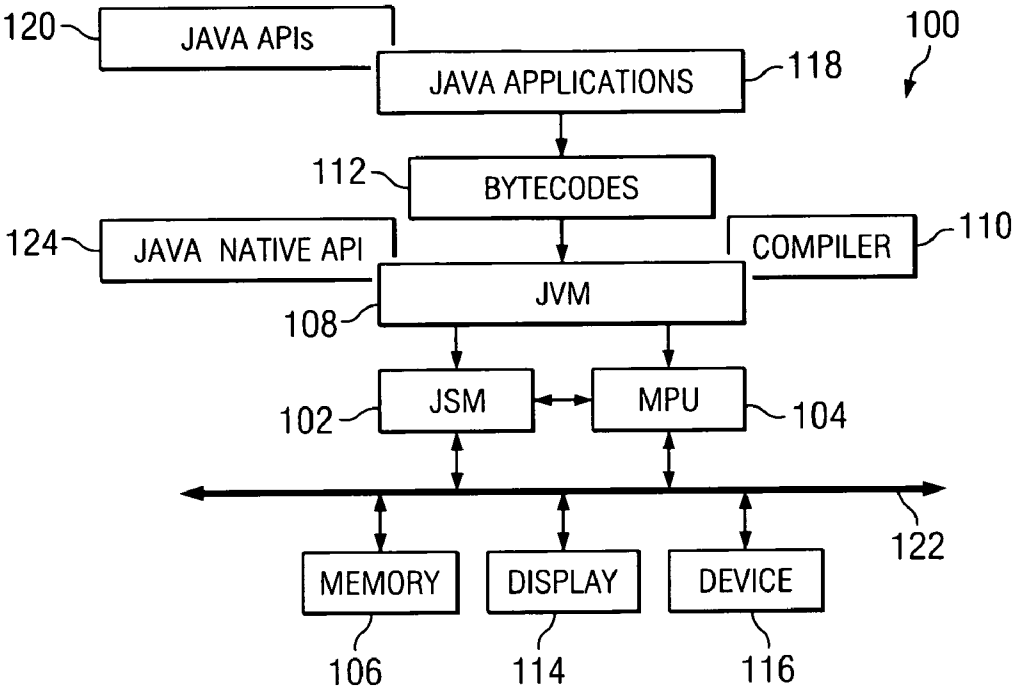
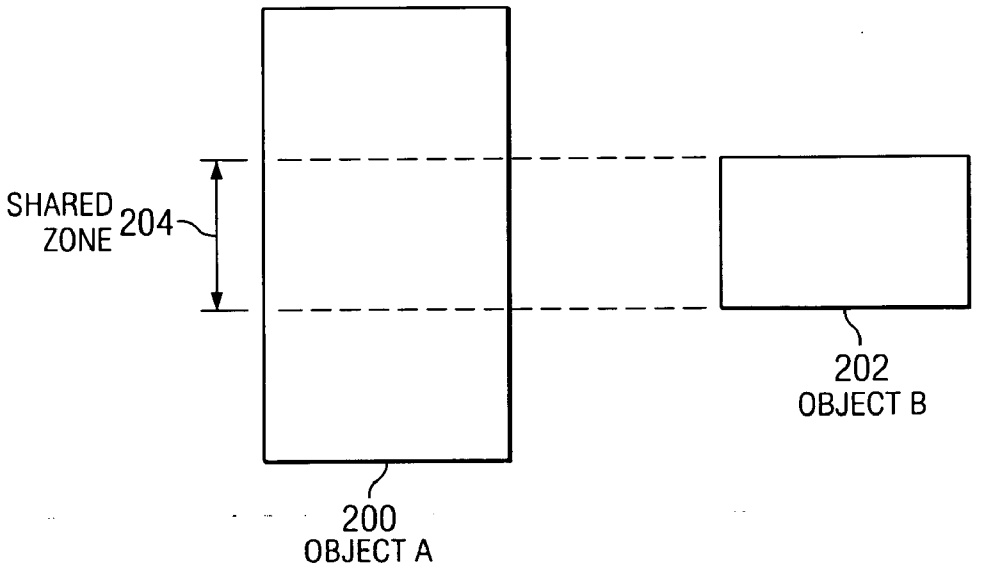

120
JAVA APIs
100
JAVA APPLICATIONS
118
112
BYTECODES
124
JAVA NATIVE API
COMPILER
110
JVM
108
102
JSM
MPU
104
122
MEMORY
DISPLAY
DEVICE
106
114
116

*FIG. 2*


SHARED ZONE
204
202
OBJECT B
200
OBJECT A

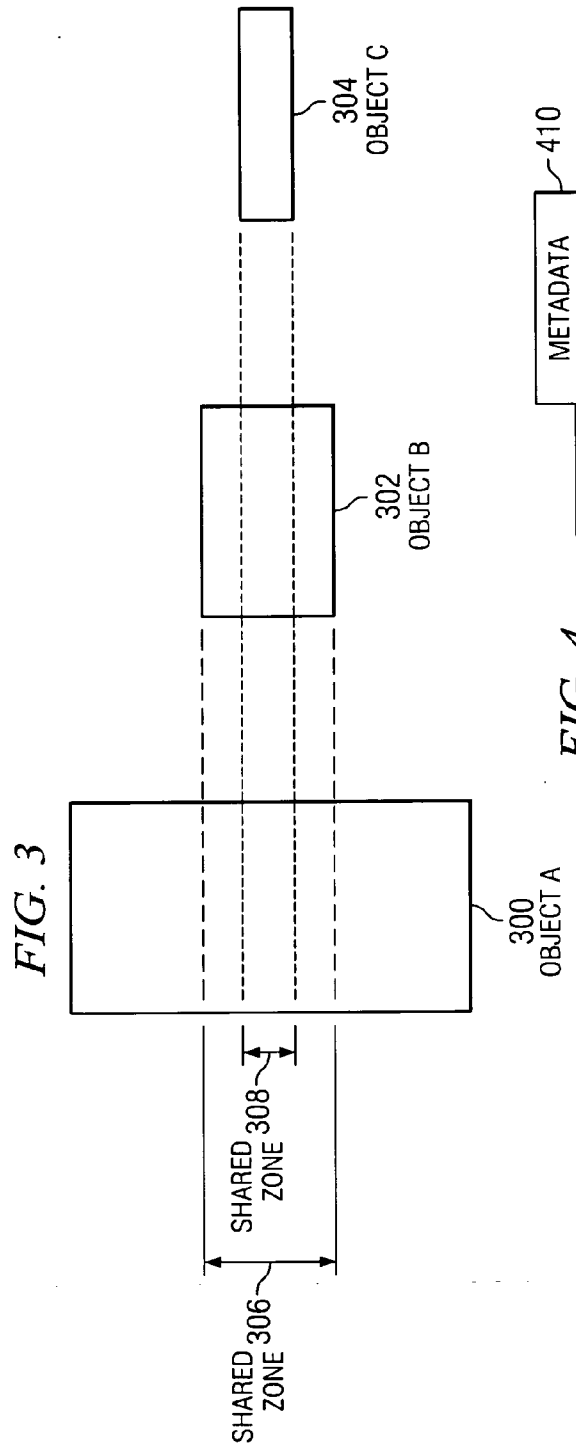

304
OBJECT C
302
OBJECT B
300
OBJECT A
SHARED ZONE 308
SHARED ZONE 306

FIG. 3

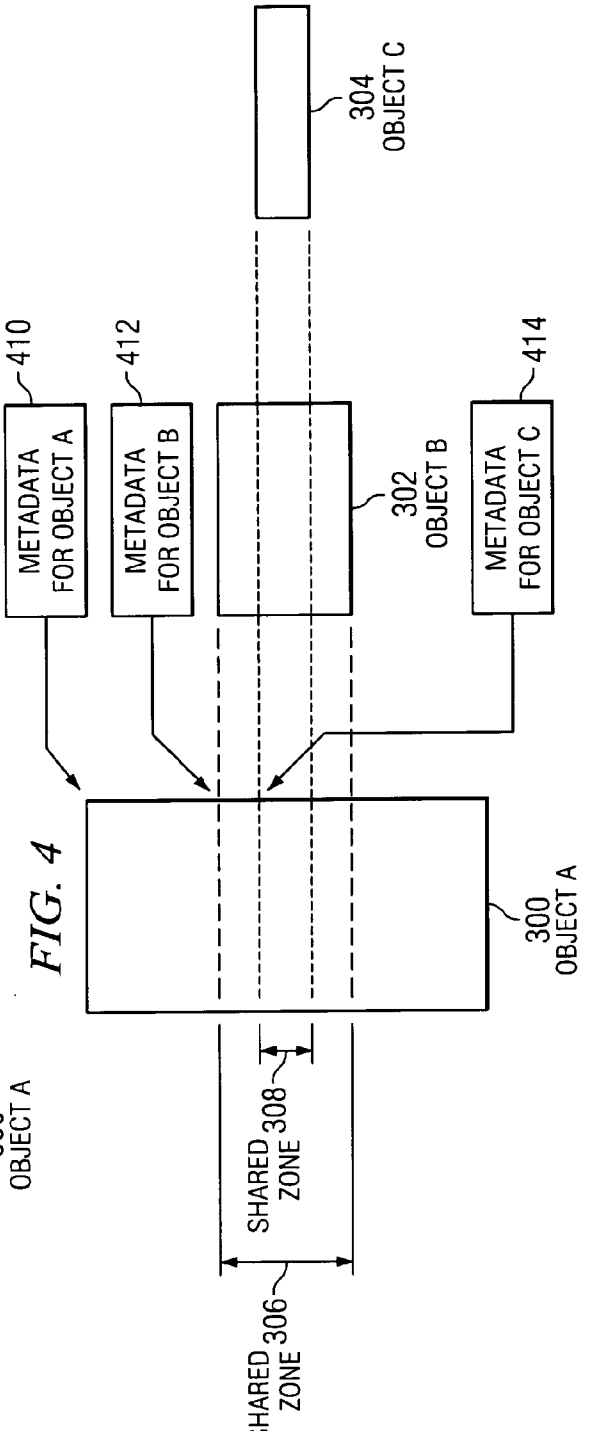

304
OBJECT C
410
METADATA FOR OBJECT A
412
METADATA FOR OBJECT B
414
METADATA FOR OBJECT C
302
OBJECT B
300
OBJECT A
SHARED ZONE 308
SHARED ZONE 306

FIG. 4

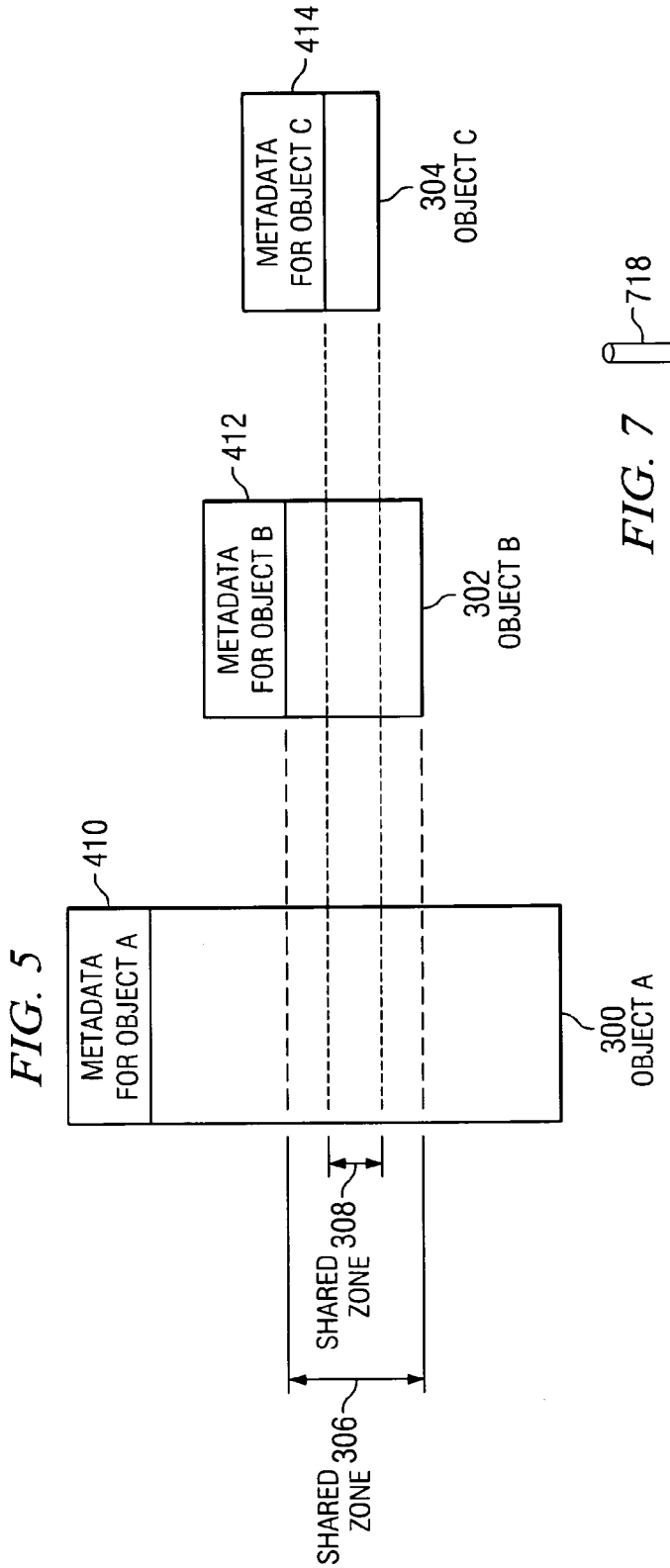

METADATA FOR OBJECT A
410
300 OBJECT A
METADATA FOR OBJECT B
412
302 OBJECT B
METADATA FOR OBJECT C
414
304 OBJECT C
SHARED ZONE 308
SHARED ZONE 306

FIG. 5

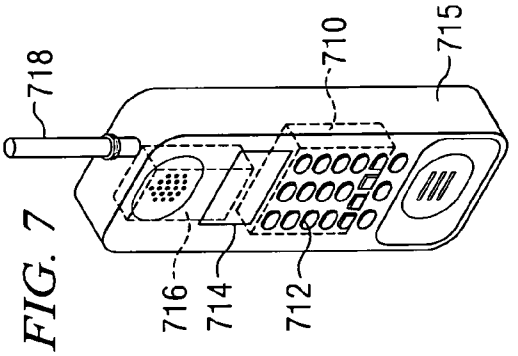

718
710
715
716
714
712

FIG. 7

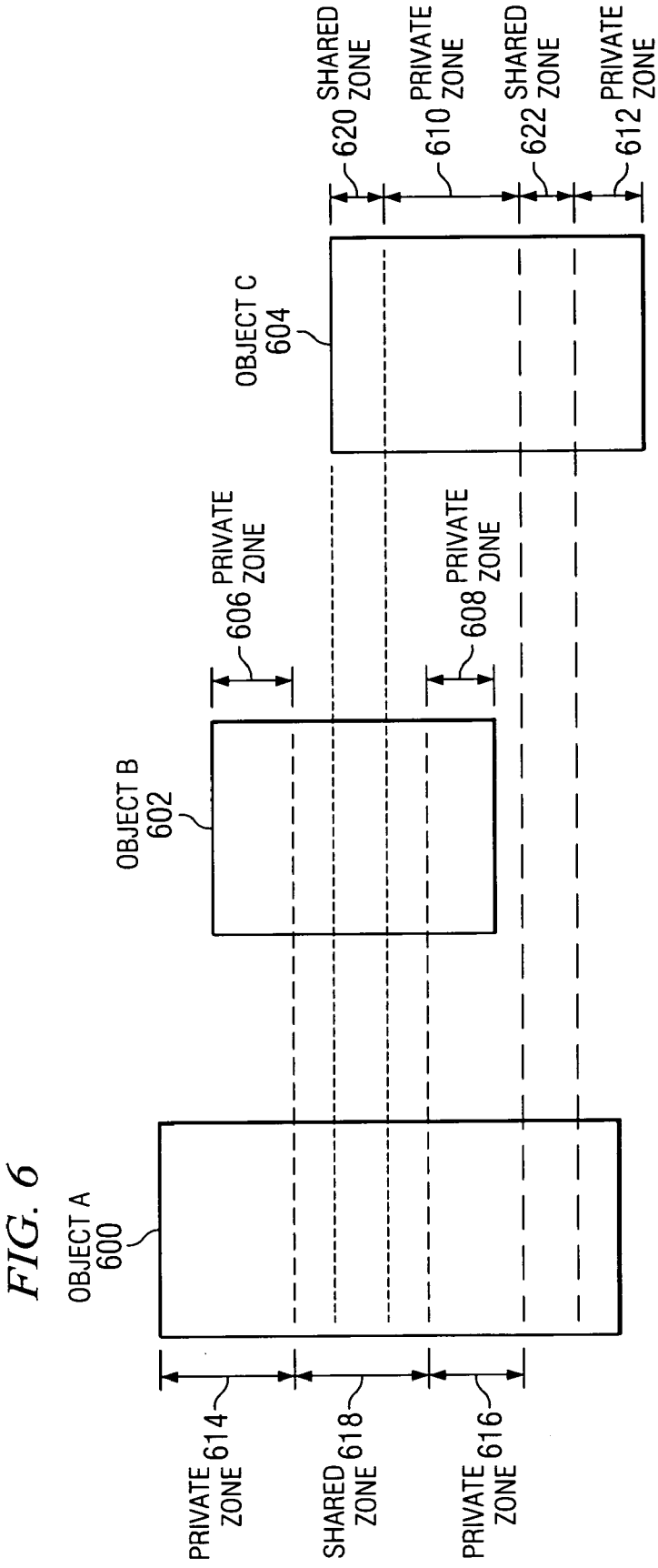

OBJECT A
600
OBJECT B
602
OBJECT C
604
PRIVATE ZONE
614
SHARED ZONE
618
PRIVATE ZONE
616
PRIVATE ZONE
606
PRIVATE ZONE
608
SHARED ZONE
620
PRIVATE ZONE
610
SHARED ZONE
622
PRIVATE ZONE
612

FIG. 6

# METHOD AND SYSTEM FOR SHARED OBJECT DATA MEMBER ZONES

## CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of European Patent Application No. 04291918.3, filed Jul. 27, 2004, incorporated by reference herein as if reproduced in full below.

## BACKGROUND OF THE INVENTION

[0002] In software applications written in object-oriented languages such as Java™, C++, and Smalltalk, objects are generally allocated as individual entities. That is, each object has its own memory space with private data that is not shared with or accessible by any other object. However, there are many applications that duplicate subsets of data between objects, such as network protocol management, multimedia decoding, sound generation, network packet encoding and decoding, and audio/video streaming. Enhancements to improve sharing of duplicate data between objects in object-oriented applications are desirable.

## SUMMARY

[0003] Accordingly, there are disclosed herein methods, computer-readable media, and systems for sharing duplicate data between objects in object-oriented applications. Some embodiments provide a method for sharing data member zones of objects in a software application executing on a processor. The method includes instantiating a first object comprising a first plurality of data members and instantiating a second object comprising a second plurality of data members. The method further includes defining a first shared data member zone comprising a first portion of the first plurality of data members and a second portion of the second plurality of data members, and modifying a value of a data member in the first portion, the modification making the value available to a read access of a corresponding data member in the second portion.

[0004] Some embodiments provide a computer-readable medium that stores a software program that when executed by a processor performs the above-described method. Other embodiments provide a system that comprises a processor, an implementation of an object-oriented language that executes on the processor, and a software program that executes on the implementation of the object-oriented language. The software program is configured to perform the above-described method.

## NOTATION AND NOMENCLATURE

[0005] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, semiconductor companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to . . . ". Also, the term "couple" or "couples" is intended to mean either an indirect or direct connection. Thus, if a first device couples to a second device, that connection may be through a direct connection, or through an indirect connection via other devices and connections.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0006] For a more detailed description of the preferred embodiments of the present invention, reference will now be made to the accompanying drawings, wherein:

[0007] **FIG. 1** shows a diagram of a system in accordance with embodiments of the invention;

[0008] **FIGS. 2-6** illustrate methods for shared object data zones in accordance with embodiments of the invention; and

[0009] **FIG. 7** depicts an illustrative embodiment of the system described herein.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0010] The following discussion is directed to various embodiments of the invention. Although one or more of these embodiments may be preferred, the embodiments disclosed should not be interpreted, or otherwise used, as limiting the scope of the disclosure, unless otherwise specified. In addition, one skilled in the art will understand that the following description has broad application, and the discussion of any embodiments is meant only to be exemplary of those embodiments, and not intended to intimate that the scope of the disclosure, is limited to those embodiments.

[0011] The subject matter disclosed herein is directed to methods that provide shared object data zones in object-oriented languages. In embodiments of these methods, two or more objects may share one or more data zones. After the objects are instantiated in memory, a method provided by a virtual machine or run time implementing the language is called to define the shared zones between pairs of objects. Once the shared zones are defined, the virtual machine is responsible for maintaining consistency between the shared zones, if such consistency maintenance is needed. Merely by way of example, the embodiments described herein are directed to Java and a Java Virtual Machine implemented on a Java processor referred to herein as a Java Stack Machine. These embodiments should not be construed as limitations of the scope of this disclosure. The methods described herein are applicable to other object-oriented languages such as C++ and Smalltalk, and to other processors including general purpose processors.

[0012] **FIG. 1** shows a system **100** in accordance with embodiments of the invention. As shown, the system **100** may comprise at least two processors **102** and **104**. Processor **102** may be referred to for purposes of this disclosure as a Java Stack Machine ("JSM") and processor **104** may be referred to as a Main Processor Unit ("MPU"). The system **100** may also comprise memory **106**, and a display **114** coupled to both the JSM **102** and MPU **104** via one or more busses **122**. At least a portion of the memory **106** may be shared by both processors, and if desired, other portions of the memory **106** may be designated as private to one processor or the other. Other components such as disk drives and controllers (not specifically shown) may be included as desired for various applications.

[0013] The system **100** also comprises a Java Virtual Machine ("JVM") **108**, compiler **110**, Java APIs **120**, Java

native APIs **124**, and Java applications **118**. The JVM may comprise a class loader, bytecode verifier, garbage collector, and a bytecode interpreter loop to interpret the bytecodes that are not executed on the JSM processor **102**. The Java applications **118** are written in Java language source code and may comprise references to one or more classes of the Java Application Program Interfaces ("APIs") **120** and the Java native APIs **124**. The Java native APIs **124** comprises interfaces to classes and methods implemented in other languages such as C++, C or assembler.

[**0014**] The Java source code is converted or compiled to a series of bytecodes **112**, with each individual one of the bytecodes referred to as an "opcode." Bytecodes **112** are provided to the JVM **108**, possibly compiled by compiler **110**, and provided to the JSM **102** and/or MPU **104** for execution. In some embodiments, the JSM **102** may execute at least some Java bytecodes directly. When appropriate, however, the JVM **108** may also request the MPU **104** to execute one or more Java bytecodes not executed or executable by the JSM **102**. In addition to executing compiled Java bytecodes, the MPU **104** also may execute non-Java instructions.

[**0015**] The MPU **104** may also host an operating system ("O/S")(not specifically shown) which performs various functions such as system memory management, the system task management that schedules the software aspects of the JVM **108** and most or all other native tasks running on the system, management of the display **114**, and receiving input from input devices (e.g., device **116**). Java code may be used to perform any one of a variety of applications such as multimedia, games or web based applications in the system **100**, while non-Java code, which may comprise the O/S and other native applications, may still run on the system on the MPU **104**.

[**0016**] Java bytecodes perform stack-based operations. For example, an "IADD" (integer add) Java opcode pops two integers off the top of the stack, adds them together, and pushes the sum back on the stack. A "simple" opcode is one in which the JSM **102** may perform an immediate operation either in a single cycle (e.g., an IADD opcode) or in several cycles (e.g., "DUP2_X2"). A "complex" opcode is one in which several memory accesses may be required to be made within the JVM data structure for various verifications (e.g., NULL pointer, array boundaries).

[**0017**] A JSM processor **102** in accordance with embodiments of the invention may execute, in addition to the Java bytecodes, a second instruction set other than Java™ bytecodes. In some embodiments, the second instruction set may comprise register-based and memory-based operations rather than stack-based operations. This second instruction set complements the Java instruction set and, accordingly, may be referred to as a complementary instruction set architecture ("C-ISA"). By complementary, it is meant that some complex Java bytecodes may be replaced by a "micro-sequence" comprising C-ISA instructions. The execution of Java™ code may thus be made more efficient and run faster by replacing some opcodes with more efficient micro-sequences of C-ISA instructions. For example, the compiler **110** may scan a series of Java bytes codes and replace one or more of such bytecodes with an optimized code segment mixing CISA and bytecodes and which is capable of more efficiently performing the function(s) performed by the initial group of Java bytecodes. In at least this way, Java execution may be accelerated by the JSM **102**.

[**0018**] In some embodiments, the JVM **108** provides a method for sharing data member zones between two or more Java objects. As used herein, a data member is a field or instance variable of a Java class, and a data member zone is one or more data members allocated in contiguous memory locations in a Java object. Each Java object may have its own view of the shared data member zone. There is no requirement that the names or types of the data members in the shared data member zone be the same in each object. The only requirement is that the shared data member zone must be the same physical size in each object. For example, as illustrated in **FIG. 2**, Object A **200** and Object B **202** may share a data member zone **204**. Object A **200** may define the shared data member zone **204** as an array of four integers and those data members are accessed using array operations. Object B **202** may define the shared data member zone **204** as four integer fields and those data members are accessed using Java field operations.

[**0019**] In some embodiments, the JVM **108** comprises a Java native method, referred to herein as ShareObjectZone, that is used to define data member zones shared between two Java objects. This Java native method comprises five parameters: the two objects, e.g., object A **200** and object B **202**, which share a data member zone **204**, the base of the shared data member zone of object A **200**, the base of the shared data member zone of object B **202**, and the size of the shared data member zone **204**. In at least one embodiment, the call to ShareObjectZone must be done after the sharing objects are instantiated and before any of the sharing objects are used. As will be explained in more detail in reference to **FIGS. 4 and 5** below, once the shared data member zone **204** is defined, any changes made to the shared data member zone **204** of Object A **200** may be automatically made available in the shared data member zone of Object B **202** and vice versa.

[**0020**] Data member zones may be shared between two or more objects and may be layered. For example, as illustrated in **FIG. 3**, Object A **300** and Object B **302** share a data member zone **306**, and Object A **300**, Object B **302**, and Object C **304** all share a data member zone **308**. Note that shared data member zone **308** is part of shared data member zone **306**, and any changes made in shared data member zone **308** are automatically made in shared data member zone **306**. Any changes made to the shared data member zone **308** in any one of three objects **300**, **302**, **304** may be automatically made available in the shared data member zones of the other two objects. In some embodiments, multiple calls to ShareObjectZone are used to define the sharing of data member zone **306** and data member zone **308** between the pairs of objects.

[**0021**] The JVM **108** performs any necessary operations to maintain consistency between objects sharing data member zones. How this consistency is maintained depends on the internal object representation used by the JVM **108**, and in some embodiments, a consistency policy defined for the sharing objects. In the JVM **108**, each Java object is comprised of a small fixed-sized group of metadata fields (sometimes referred to as a header), and some number of instance fields, i.e., data members. The data members of an object are allocated contiguously in memory and may be

accessed via fixed offsets from the beginning of the contiguous memory space. In some embodiments, the metadata for an object is allocated in a separate area of memory apart from the data members. In such embodiments, the metadata comprises a reference or pointer to the memory containing the data members. In other embodiments, the metadata for an object is allocated in memory contiguously with the data members, either preceding or following the data members, and no pointer is needed in the metadata.

[0022] **FIG. 4** illustrates a method for sharing data member zones between objects when the metadata for an object is allocated in a separate area of memory. In embodiments such as that illustrated by **FIG. 4**, the JVM **108** allocates the metadata **410**, **412**, **414**, for Object A **300**, Object B **302**, and Object C **304** in memory apart from the memory containing the data members of the objects. Each of the metadata blocks **410**, **412**, **414** contains a pointer or reference to the block of memory containing the data members of the object. To create the shared data member zone **306** between Object A **300** and Object B **302**, the pointer to the data member memory of Object B **302** contained in the metadata of Object B **412** is changed to point to the beginning of the shared data member zone **306** in the memory allocated for Object A **300**. Similarly, to define the shared memory data zone **308**, the pointer to the data member memory of Object C **304** contained in the metadata of Object C **414** is changed to point to the beginning of the shared data member zone **308** in the memory allocated for Object A **300**.

[0023] In these embodiments, the shared data member zones **306** and **308** are in the same physical memory and no additional work is required of the JVM **108** to maintain consistency between the shared data members of the three objects. For example, if a data member of shared data member zone **308** is changed by a Java field access in Object C **304**, the change is actually made in the physical memory of Object A **300**. If a read of the corresponding data member of the shared data member zone **308** is performed by a Java field access in Object B **302**, the value is read from the physical memory allocated to Object A **300**.

[0024] In addition, in these embodiments, the garbage collector may need to be augmented to recognize that multiple objects have references to a shared data member zone. For example, since Object B **302** and Object C **304** access the physical memory allocated to Object A **300**, the garbage collector should not free up the memory allocated to Object A **300** unless all three objects are no longer live. Therefore, if Object A **300** is reachable from the root context of the garbage collector, Object B **302** and Object **304** must also be reachable from that root context. In some embodiments, the JVM **108** may maintain a simple table of the shared data member zones to be used by the garbage collector.

[0025] **FIG. 5** illustrates a method for sharing data member zones between objects when the metadata for an object is allocated contiguously with the data members of the object. In embodiments such as that illustrated by **FIG. 5**, the JVM **108** allocates the metadata **410**, **412**, **414**, for Object A **300**, Object B **302** and Object C **304** in memory contiguous to the data members of the objects. Object A **300** and Object B **302** each have separate physical memory representations of the shared data memory zone **306**, and Object A **300**, Object B **302**, and Object C **304** each have separate physical memory representations of the shared data memory zone **308**. In these embodiments, the JVM **108** is configured to maintain consistency between these separate physical memory representations. In some embodiments, each time a data member in shared data member zone **306** or shared data member zone **308** is modified by an access in one of the sharing objects, the JVM **108** automatically copies, i.e., propagates, the modification to the corresponding data members in the other sharing objects. For example, if a data member of shared data member zone **308** is changed by a Java field access in Object C **304**, the JVM **108** also changes the corresponding data member in Object B **302** and Object A **300**.

[0026] In various embodiments, the JVM **108** provides the capability to specify consistency attributes for shared data member zones rather than automatically performing copies each time a data member in a shared data member zone is changed. In such embodiments, when a shared data member zone **306** or **308** is defined, consistency attributes may also be defined to indicate which changes should be propagated automatically. In some embodiments, a consistency attribute may be set for an entire shared data member zone. In other embodiments, a consistency attribute may be set for each data member of a shared data member zone. Other embodiments allow, on a per shared data member zone basis, setting a consistency attribute for the entire shared data member zone or for individual data members within the shared data member zone. In any of these embodiments, the consistency attribute may be set to indicate whether a change to the shared data member zone or to the shared data member is to be propagated automatically when the change is made or not.

[0027] Some embodiments also comprise a native method, e.g., MakeMySharedZoneConsistent, in the JVM **108** that may be called by a Java application to propagate changes made to a shared data member zone and/or to individual data members in a shared data member zone as needed. In at least one embodiment, this native method comprises two parameters, a reference to an object sharing a shared data member zone, and a reference to the base of the shared data member zone in that object to be updated. The native method causes any changes made in the corresponding shared data member zones of other objects to be propagated to the shared data member zone of the referenced object.

[0028] A Java application may use consistency attributes and/or the Java native method to set up a consistency policy that matches the application's use of a shared data member zone. For example, when shared data member zone **308** is defined, the defining Java application may specify that all modifications made to the shared data member zone **308** made in Object C **304** should be automatically propagated to the shared zones in Object A **300** and Object B **302**, but no changes made in the shared data memory zone **308** in Object A **300** should be automatically propagated to the shared zones in Object B **302** or Object C **304**. If the Java application periodically needs to have the changes made in the shared data memory zone **308** in Object A **300** propagated to the other sharing objects, the Java application may invoke the Java native method as needed to perform the copying.

[0029] In the previous embodiments described herein, the shared data member zones of the objects have been nested,

i.e. all data members of one member of each sharing pair of objects are shared with the other member of the pair. For example, all of the data members of Object C **304** are in the shared data member zone **308**, and all of the data members of Object B **302** are in shared data member zone **306**. **FIG. 6** illustrates a method for sharing one or more subsets of data members between objects while keeping other subsets private. As shown in **FIG. 6**, a shared data member zone **618** is defined between Object A **600** and Object B **602**, a shared data member zone **622** is defined between Object A **600** and Object C **604**, and a shared data member zone **620** is defined between Object A **600**, Object B **602**, and Object C **604**. Object A **600** has private data member zones **614** and **616**, Object B has private data member zones **606** and **608**, and Object C has private data member zones **610** and **612**. In some embodiments, multiple invocations of ShareObjectZone are used to create the various shared data member zones **618**, **620**, **622**.

[**0030**] The JVM **108** is configured to recognize which subsets of the object data members are private and which are shared. In some embodiments, when a data member in a shared data member zone is changed in one of the sharing objects, the JVM **108** automatically propagates the change to the corresponding shared zone in any other sharing objects. For example, if a data member in the shared data member zone **620** is changed in Object C **604**, the JVM **108** automatically makes the change in the corresponding data members in Object A **600** and Object B **602**. In other embodiments, the JVM **108** provides the capability to specify consistency attributes for shared data member zones rather than automatically performing copies each time a data member in a shared data member zone is changed. Such consistency attributes are described in more detail in reference to **FIG. 5**.

[**0031**] System **100** may be implemented as a mobile device **715** such as that shown in **FIG. 7**. As shown, the mobile device **715** includes an integrated keypad **712** and display **714**. The JSM processor **102** and MPU processor **104** and other components may be included in electronics package **710** connected to the keypad **712**, display **714**, and radio frequency ("RF") circuitry **716**. The RF circuitry **716** may be connected to an antenna **718**.

[**0032**] While the various embodiments of the invention have been shown and described, modifications thereof can be made by one skilled in the art without departing from the spirit and teachings of the invention. The embodiments described herein are illustrative only, and are not intended to be limiting. Many variations and modifications of the invention disclosed herein are possible and are within the scope of the invention. Accordingly, the scope of protection is not limited by the description set out above. Each and every claim is incorporated into the specification as an embodiment of the present invention.

What is claimed is:

**1**. A method for sharing data member zones of objects in a software application executing on a processor, the method comprising:

instantiating a first object comprising a first plurality of data members;

instantiating a second object comprising a second plurality of data members;

defining a first shared data member zone comprising a first portion of the first plurality of data members and a second portion of the second plurality of data members; and

modifying a value of a data member in the first portion, the modification making the value available to a read access of a corresponding data member in the second portion.

**2**. The method of claim 1, wherein modifying a value further comprises propagating the value to a physical memory location of the corresponding data member.

**3**. The method of claim 2, wherein

defining a first shared data member zone further comprises setting a consistency attribute of the first portion to indicate whether modifications of data members in the first portion should be propagated, and

propagating the value further comprises propagating the value if the consistency attribute is set to indicate that modifications should be propagated.

**4**. The method of claim 2, wherein

defining a first shared data member zone further comprises setting a consistency attribute of the data member in the first portion to indicate whether modifications to the data member should be propagated, and

propagating the value further comprises propagating the value if the consistency attribute of the data member is set to indicate that modifications should be propagated.

**5**. The method of claim 1, wherein

defining a first shared data member zone further comprises configuring the first object to cause an access of the data member in the first portion to access a physical memory location of the corresponding data member in the second portion, and

modifying a value further comprises placing the value in the physical memory location.

**6**. The method of claim 5, wherein configuring the first object further comprises causing a pointer field in a header of the first object to point to the second portion.

**7**. The method of claim 1, further comprising:

instantiating a third object comprising a third plurality of data members;

defining a second shared data member zone comprising a third portion of the third plurality of data members, a subset of the first portion, and a subset of the second portion, wherein the third plurality and the subsets are the same size; and

modifying a value of a data member in the third portion, the modification making the value available to a read access of a corresponding data member in the subset of the first portion and a corresponding data member in the subset of the second portion.

**8**. The method of claim 7, wherein modifying a value of a data member in the third portion further comprises propagating the value to a physical memory location of the corresponding data member in the subset of the first portion and a physical memory location of the corresponding data member in the subset of the second portion.

9. The method of claim 8, wherein

defining a second shared data member zone further comprises setting a consistency attribute of the third portion to indicate whether modifications of data members in the third portion should be propagated, and

propagating the value to the physical memory locations of the corresponding data members in the subsets of the first and second portions further comprises propagating the value if the consistency attribute is set to indicate that modifications should be propagated.

10. The method of claim 7, wherein

defining a second shared data member zone further comprises configuring the third object to cause an access of the data member in the third portion to access a physical memory location of the corresponding data member in the subset of the second portion, and

modifying a value of a data member in the third portion further comprises placing the value in the physical memory location of the corresponding data member in the subset of the second portion.

11. The method of claim 10, wherein configuring the third object further comprises causing a pointer field in a header of the third object to point to the subset of the second portion.

12. The method of claim 1, wherein the first object and the second object are Java objects.

13. A computer-readable medium storing a software program that, when executed by a processor, performs a method for sharing data member zones of objects comprising:

instantiating a first object comprising a first plurality of data members;

instantiating a second object comprising a second plurality of data members;

defining a first shared data member zone comprising a first portion of the first plurality of data members and a second portion of the second plurality of data members; and

modifying a value of a data member in the first portion, the modification making the value available to a read access of a corresponding data member in the second portion.

14. The computer-readable medium of claim 13, wherein modifying a value further comprises propagating the value to a physical memory location of the corresponding data member.

15. The computer-readable medium of claim 14, wherein

defining a first shared data member zone further comprises setting a consistency attribute of the first portion to indicate whether modifications of data members in the first portion should be propagated, and

propagating the value further comprises propagating the value if the consistency attribute is set to indicate that modifications should be propagated.

16. The method of claim 14, wherein

defining a first shared data member zone further comprises setting a consistency attribute of the data member in the first portion to indicate whether modifications to the data member should be propagated, and

propagating the value further comprises propagating the value if the consistency attribute of the data member is set to indicate that modifications should be propagated.

17. The computer-readable medium of claim 13, wherein

defining a first shared data member zone further comprises configuring the first object to cause an access of the data member in the first portion to access a physical memory location of the corresponding data member in the second portion, and

modifying a value further comprises placing the value in the physical memory location.

18. The computer-readable medium of claim 17, wherein configuring the first object further comprises causing a pointer field in a header of the first object to point to the second portion.

19. The computer-readable medium of claim 13, wherein the first object and the second object are Java objects.

20. A system, comprising:

a processor;

an implementation of an object oriented language configured to execute on the processor; and

a software program configured to execute on the implementation of the object-orientated language, wherein the software program is configured

to instantiate a first object comprising a first plurality of data members,

to instantiate a second object comprising a second plurality of data members,

to define a first shared data member zone comprising a first portion of the first plurality of data members and a second portion of the second plurality of data members, and

to modify a value of a data member in the first portion, the modification making the value available to a read access of a corresponding data member in the second portion.

21. The system of claim 20, wherein the software program is further configured to modify a value by propagating the value to a physical memory location of the corresponding data member.

22. The system of claim 21, wherein the software program is further configured to define a first shared data member zone by setting a consistency attribute of the first portion to indicate whether modifications of data members in the first portion should be propagated, and propagating the value further comprises propagating the value if the consistency attribute is set to indicate that modifications should be propagated.

23. The system of claim 21, wherein the software program is further configured to define a first shared data member zone by setting a consistency attribute of the data member in the first portion to indicate whether modifications to the data member should be propagated, and propagating the value further comprises propagating the value if the consistency attribute of the data member is set to indicate that modifications should be propagated.

**24**. The system of claim 20, wherein the software program is further configured

to define a first shared data member zone by configuring the first object to cause an access of the data member in the first portion to access a physical memory location of the corresponding data member in the second portion, and

to modify a value by placing the value in the physical memory location.

**25**. The system of claim 24, wherein configuring the first object further comprises causing a pointer field in a header of the first object to point to the second portion.

**26**. The system of claim 20, wherein the implementation of the object-oriented language comprises a Java virtual machine.

**27**. The system of claim 20, wherein the system comprises a mobile device.

* * * * *