UNIVERSITETET I AGDER

FAKULTET FOR ØKONOMI OG SAMFUNNSVITENSKAP

E K S A M E N

Emnekode:	**IS-207**
Emnenavn:	Algorithms and Data Structures
Dato:	22 May 2015
Varighet:	**0900-1300**
Antall sider inkl. forside:	4
Målform:	**English**
Tillatte hjelpemidler:	**Any books or notes**

Background

In Java you can write programs that create new objects, and discard them, without worrying about running out of memory.
Programming languages with automatic memory allocation, like Java, must be able to detect and reuse the parts of the memory that the program cannot use any more. This process is usually called garbage collection. In this problem set you will work with a couple of garbage collection algoithms. The first one is known as naïve mark and sweep, the other is often called the LISP2 algorithm.

The mark and sweep garbage collector

Memory for objects are allocated from a part of memory known as the heap. The garbage collector maintains a linked list (the free list) of the parts of the heap that are unused. When the program creates a new object, the garbage collector traverses the free list to find a free memory block that is large enough for the object. If it finds a block of the right size and that block is removed from the freelist and used for the object. If the block is large the block will remain on the free list, but its size will be reduced so that a part is used for the object.

If the garbage collector is unable to find a memory block for the object, it start the garbage collection process to recycle memory from unused objects. When the garbage collection is finished, it will try again to allocate memory for the object.

Problem 1 Mark and sweep GC

The mark and sweep algorithm consists of two phases, mark and sweep

You will find a description of a memory model that you can use in appendix A.

.

a) The purpose of the mark phase is to find (and mark) all objects that the program can possibly use in the future. It starts from a number of known variables, any object pointed to from these variables can be used in the future, and if any reachable object has pointers (class type variables) then the objects they point to can also be reached, and so on...

   Write the mark method. You can assume that there is one variable called root. Any object that can be reached by following pointers from root is reachable and must be marked as reachable. You can assume that all objects are marked as garbage before the method is called.

b) In the sweep phase, the heap is scanned from one end to the other, any unmarked object is returned to the free list.

Problem 2 The LISP 2 garbage collector

One of the problems with the naïve mark and sweep algorithm is that the heap will become increasingly fragmented over time, making it difficult to find space for large objects. Compacting garbage collectors, e.g. LISP2, solve this problem by moving all the usable

objects to one end of the heap, leaving one big area of free memory to allocate objects from. An additional advantage is that the free list is not needed anymore, only the address to the start of the block of free memory.

We still need to find the usable objects, so the mark phase is the same as in mark and sweep, but the sweep phase is not used. Instead all objects will be moved as far as possible towards the start of the heap, which means that their addresses will change, and consequently that all variables that point to the object must also be changed. This is done in two phases:

a) The first phases calculates the new address of each usable object, and stores the new address in a "secret" newAddress field in the object, which is only used for garbage collection. This is found by scanning the heap from start to end, and adding up the sizes of the usable objects, to keep track of how much memory is used so far. Whenever a usable object is encountered the current sum will be its new address.

   Write the calculateAdrresses() method.

b) The second phase is to update the pointers. Now every object has its new addressed stored in the newAddress field. To update the pointers the heap must be scanned again. In usable objects the pointers are updated by changing the pointer to the value of the newAddress field in the object they point to

   Write the updatePointers() method

c) The final phase is to move the objects. They must be copied byte for byte to their new address.

   Write the moveObjects() method.

Problem 3 Discussion

For each of the methods above, describe which data structures you have used, and why. You may also comment on the structures that were given in the problem set.

You should also consider alternatives. Could, for instance, any of these have been used in stead: List, Stack, Queue, Map…

Appendix A

Memory model.

Insert appropriate code from the Heap class here….