UNIVERSITETET I AGDER

E K S A M E N

Emnekode:	**IS-207**
Emnenavn:	**Algorithms and Data Structures**
Dato:	**27. May 2009**
Varighet:	**0900-1300**
Antall sider inkl. forside:	5
Målform:	**English**
Tillatte hjelpemidler:	**All printed matter and handwritten notes are allowed**
Merknader:	Read the whole problem set (including appendices) before you start to write your answer. You may use any classes you want from the textbook or the java standard library in your answers to the programming problems.

UNIVERSITETET I AGDER
SERVICEBOKS 422 4604 KRISTIANSAND
TELEFON 38 14 10 00 FAKS 38 14 10 01
ORG.NR. 970 546 200 MVA postmottak@uia.no www.uia.no

Introduction

Shipping is a considerable source of air pollution. It has been estimated that up to 40% of the air pollution in Europe could come from ships in 2010. In an attempt to reduce the pollution from ships some ports plan to introduce scheduling systems. In this problem set we will work with such a system.

Normally ships are loaded or unloaded in the same order as they arrive at the port. If the port is full any new arrivals will have to wait for their turn outside the port. On their way to busy ports ships often try to get ahead of the queue by running at higher speed. Other ships will respond in the same way and they end up racing at full speed towards the port.

A ship burns a lot more fuel per mile at full speed than they do at their most economical speed. The pollution from the ship is proportional to the fuel spent, so it would be beneficial to the environment, as well as the economy of the ship, if it had an incentive to run at economy speed. This is where the scheduling system fits in. The system keeps track of all ships underway to the port and reserves a berth for each ship at the time it will arrive if it is running at economy speed. Then there is nothing to gain by running at higher speed. The ship will just have to wait if it arrives early.

We will disregard some of the issues that complicate the real system. We make the assumption that a ship leaves the port every hour, making room for one new ship. The ship should arrive exactly on the hour.

Problem 1 – Simplified scheduling

A simple solution is to schedule the ships which are on their way to the port now, and assign each of them to an arrival slot. The arrival slot should be the earliest free slot that the ship can reach without exceeding its economy speed.

a) Explain which data structure(s) you will use to solve this problem, and why you chose it/them.
b) Write a method that takes a list or an array of Ship objects as input, and assigns a unique target arrival time to each ship. To arrive at this time the ship should not need to sail faster than its economy speed. You can assume that time is measured as hours into the future. (Now is zero, and days don't exist).

UNIVERSITETET I AGDER
SERVICEBOKS 422 4604 KRISTIANSAND
TELEFON 38 14 10 00 FAKS 38 14 10 01
ORG.NR. 970 546 200 MVA postmottak@uia.no www.uia.no

Problem 2

Now we will try to deal with more of the real life complexities:

- We have a mix of ships which have already been assigned an arrival slot, and new ships which have just reported that they are under way to the port.
    - As many (most?) of the slots are assigned to a ship it may be difficult to find a slot for a new ship. Some of the possible strategies are:
      The easiest is to assign the ship to the next available slot. If this is far into the future the ship may have to sail at a very slow speed.
    - To avoid this you may reschedule some of the "old" ships to make room for the new one. They will not like it so you should only do this if the new ship would have to sail at less than half of its economy speed otherwise.
    - An alternative is to allow the new ship to speed up, if there is a vacant slot that the ship can reach by sailing up to 20% faster than economy speed.
- Note: You do not have to use all of these. The primary priority is still to ensure that (at most) one ship arrives every hour, without exceeding its economy speed.

a) Which data structure(s) will you use now? Explain why?

b) Write a class Port which keeps track of all the incoming ships, and which slot they have been assigned to. The class contains fields for data structures you chose in part a, and a method for updating the schedule as described above.
   You can still assume that time is measured as hours into the future.

Appendix 1: Speed–Time–Distance Relationships

Speed is measured in knots, time in hours, and distances in nautical miles.
Then

- speed = distance / time
- time = distance / speed
- distance = time * speed

Appendix 2: class Ship

/** You may use this class and add fields and methods as needed */

```
public class Ship {
    /** Unique id, e.g. name or callsign */
    public String id;

    /** Max economical speed in knots */
    public double economySpeed;

    /** Distance from the port in nautical miles
      * You can assume this is set automatically,
      * e.g. by a satellite surveillance system. */
    public double distance;

    /** Arrival time assigned by the system */
    public int targetArrivalTime;
}
```

Appendix 3: Java standard libray classes

java.util
Interface List<*E*>:

boolean **add**(*E* e) - Appends the specified element to the end of this list
void **add**(int index, *E* e) - Inserts the specified element at the specified position in this list
E **get**(int index) - Returns the element at the specified position in this list.
E **remove**(int index) - Removes the element at the specified position in this list
E **set**(int index, E element) - Replaces the element at the specified position in this list with the specified element
int **size**()

Implementations:
java.util.ArrayList
java.util.LinkedList

java.util
Interface Map<*K*,*V*>

boolean **containsKey**(Object key) - Returns true if this map contains a mapping for the specified key
V **get**(Object key) - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V **put**(*K* key, *V* value) - Associates the specified value with the specified key in this map

Implementations:
java.util.HashMap
java.util.TreeMap (red-black tree)

java.util
Interface Queue<*E*>

boolean **offer**(*E* e) - Inserts the specified element into this queue
E **peek**() - Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty
E **poll**() - Retrieves and removes the head of this queue

Implementations:
java.util.LinkedList
java.util.PriorityQueue (heap)

java.lang
Class Math

static double **ceil**(double a) – rounds up to the nearest integer
static double **floor**(double a) – rounds down to the nearest integer