1 Ada select Statement

Given below are 4 code fragments A, B, C, and D calling entry Call of task T:

```ada
task T is
    entry Call;
end T;
```

Explain which statements are executed in each fragment given the following assumptions:

a. T.Call is available when the select is executed.

b. T.Call is not available when the select is executed, but does become available in the next
   i) 14 seconds.
   ii) 8 seconds.
   iii) 2 seconds.

A rendezvous with T.Call takes 5 seconds to execute.

A:

```ada
select
    T.Call;
or
    delay 10.0;
    ... -- code that takes 2 seconds to execute
end select;
```
B:
```plaintext
select
   T.Call;
else
   delay 10.0;
   ...
   -- code that takes 2 seconds to execute
end select;
```

C:
```plaintext
select
   T.Call;
then abort
   delay 10.0;
   ...
   -- code that takes 2 seconds to execute
end select;
```

D:
```plaintext
select
   delay 10.0;
   ...
then abort
   T.Call;
   ...
   -- code that takes 2 seconds to execute
end select;
```

2 Questions

Are the following statements true or false?

a. Under transient overload of a system, hard deadlines are allowed to be missed occasionally under WCET assumptions as long as they are met under ACET (average CET) assumptions.

b. Any non-determinism of timing characteristics of hard real-time applications is to be avoided at all costs.

c. When a thread terminates with an unexpected and unhandled exception, this is a “fail uncontrolled” situation.

d. Local drift is the effect that a series of delays of equal duration in a task will eventually add up to much more than the sum of the delays, because the task cannot always be scheduled immediately to run when the delay expires.

e. In Ada, parent and guardian of a task are always identical.

f. To prove that a task set scheduled with rate-monotonic priority assignment meets all its deadlines, it is sufficient to show this property for the longest period only.
3 Basic Synchronisation in Java

The following code fragments show alternative implementations of coordinates in Java. As multiple threads may concurrently access objects of type Coordinate, synchronisation is needed to prevent race conditions. In Java, methods may be declared synchronized. If one thread invokes a synchronised method on an object, the lock of that object is first acquired, the method body is executed, and then the lock is released. Another thread invoking a synchronised method on that same object will be blocked until the lock is released.

Fragment 1:
```java
class Coordinate {
    public synchronized void set(int a, int b) {x = a; y = b;}
    public synchronized int getX() {return x;}
    public synchronized int getY() {return y;}
    public synchronized boolean less(Coordinate c) {
        return getX() < c.getX() && getY() < c.getY();
    }
    private int x, y;
}
```

Fragment 2:
```java
class Coordinate {
    public synchronized void set(int a, int b) {x = a; y = b;}
    public synchronized int getX() {return x;}
    public synchronized int getY() {return y;}
    public synchronized boolean less(Coordinate c) {
        return getX() < c.getX() && getY() < c.getY();
    }
    private int x, y;
}
```

Fragment 3:
```java
class Coordinate {
    public synchronized void set(int a, int b) {x = a; y = b;}
    public synchronized int getX() {return x;}
    public synchronized int getY() {return y;}
```
public synchronized boolean less(Coordinate c) {
    return x < c.x && y < c.y;
}

private int x, y;
}

a) Analyse each alternative and identify potential problems. Explain what could go wrong at runtime. Describe potential scenarios under which an error occurs.

b) Implement a thread-safe class for coordinates that solves those problems.

4 Condition Synchronisation in Java

The following code fragment shows a class PrintBuffer that stores documents to be printed. Concurrent users may add new documents to the buffer which are processed by several printers. If a printer wants to process a new document, it will fetch it via the get method. Insert the missing statements that guarantee that multiple concurrent users and printers can safely access PrintBuffers and that the conditions hold.

class PrintBuffer {
    public void add(Document d) {
        // condition: next < N
        ...
        a[next] = d;
        next++;
        ...
    }

    public Document get() {
        Document d;
        // condition: next > 0
        ...
        next--;
        d = a[next];
        ...
        return d;
    }

    private static final int N = 10;
    private int next = 0;
    private Document a[] = new Document[N];
}

Hint: The wait-statement in Java interrupts the current thread until notify or notifyAll is invoked; notify awakens at most one thread, notifyAll notifies all waiting threads.