Segundo Certamen

1.- Haga uso de la información adjunta y desarrolle un programa en C++ que lea apellidos de teclado y los almacene en una lista. Luego se pide listar los apellidos en orden alfabético descendente.

```
#include <list>
#include <string>

void main(void)
{
  string lname;
  list <string> lnameList;
  cout << "Please enter your list of last names finishing with
`.`"<<endl;

  while(true) {
    cin >> lname;
    if (lname==".") break;
    lnameList.push_back(lname);
  }
  lnameList.sort();
  lnameList.reverse();
  for (list<string>::iterator itr=lnameList.begin();
itr!=lnameList.end();itr++)
    cout << *itr<< endl;
}
```

Alternativamente se pudo usar:

```
#include <list>
#include <string>

void main(void)
{
  string lname;
  list <string> lnameList;
  cout << "Please enter your list of last names finishing with
`.`"<<endl;

  while(true) {
    cin >> lname;
    if (lname==".") break;
    lnameList.push_back(lname);
  }
  lnameList.sort();
  for (list<string>::reverse_iterator itr=lnameList.rbegin();
itr!=lnameList.rend();itr++)
    cout << *itr<< endl;
}
```

2.- El programa listado a continuación intenta mostrar el número de veces que el mouse se ha presionado y liberado en el área de trabajo del applet.

a) Indique si el programa presenta errores de compilación.

Sí, hay errores de compilación: En la clase MouseHandler se invoca doDown() y doUp(), ambos métodos no están definidos en la clase y no se tiene alcance a los de la clase Click.

b) Indique si el programa consigue el objetivo propuesto. Si no lo consigue, haga las modificaciones necesarias para conseguirlo.

Tal cual no pasa la compilación, luego debe ser modificado de modo de tener acceso a los métodos doDown() y doUp().

La versión corregida es:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Click extends Applet {
    public void init() {
      addMouseListener(new MouseHandler(this));
    }
    public void start() {
      if (msg == null ) msg = " Counting Ups and Downs";
    }
    public void paint(Graphics g) {
      g.drawString(msg, 5, 30);
    }
    public void doDown() {
      msg = " Ups = " + u + " and Downs = " + ++d;
      repaint();
    }
    public void doUp() {
      msg = " Ups = " + ++u + " and Downs = " + d;
      repaint();
    }
    private int u=0, d=0;
    private String msg = "Counting Ups and Downs";
}

class MouseHandler extends MouseAdapter {
    private Click click;

    MouseHandler(Click app) {click=app;}
    public void mousePressed(MouseEvent e) { click.doDown(); }
    public void mouseReleased(MouseEvent e) { click.doUp(); }
}
```

c) Qué función desempeña el llamado a repaint()? ¿Qué pasaría si lo eliminamos?

Su función es actualizar la información de pantalla solicitando a la máquina virtual que efectúe un llamado a paint().

Si la omitimos, los eventos de teclas del mouse no se reflejarán en pantalla hasta que por otras razones (maximizar) se deba refrescar el applet y se invoque a paint().

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Click extends Applet {
        public void init() {
                addMouseListener(new MouseHandler());
        }
        public void start() {
                if (msg == null ) msg = " Counting Ups and Downs";
```

```
        }
        public void paint(Graphics g) {
                g.drawString(msg, 5, 30);
        }
        public void doDown() {
                msg = " Ups = " + u + " and Downs = " + ++d;
                repaint();
        }
        public void doUp() {
                msg = " Ups = " + ++u + " and Downs = " + d;
                repaint();
        }
        private int u=0, d=0;
        private String msg = "Counting Ups and Downs";
}
class MouseHandler extends MouseAdapter {
        MouseHandler() {}
        public void mousePressed(MouseEvent e) { doDown(); }
        public void mouseReleased(MouseEvent e) { doUp(); }
}
```

3.- a1)Explique por qué la siguiente secuencia de Java siempre conduce a una condición del if falsa.

```
        .....
        String s =entrada;   // entrada es un String leído previamente de teclado.
        if (s=="FIN") {.........}
        ......
```

¿Cómo debería codificarse?

Los strings son objetos en Java. Cuando se comparan objetos con ==, se arroja verdadero si los nombres siendo comparados se refieren al mismo objeto. En este caso se trata de dos objetos con eventual igual atributo, pero en todos los casos se trata de objetos distintos.

La línea se debe modificar invocando:

```
        if (s.equals("FIN")) {.........}
```

b) Explique cómo llevaría a Java el siguiente código en C++. Si no fuera posible, explique.

```
        ...
        public swap(int &a, int &b) { int t=a; a=b; b=t;}
        ....
```

No es posible, en Java no existe el paso por referencia para tipos básicos. En objetos si existe, pero en este caso la semántica de la asignación de objetos (copia no profunda) no permite lograr este resultado.

c) Dé un ejemplo de framework en Java.

La clase Applet

4.- Considere la siguiente clase en Java:

```
public class myclass{
        public void setProbabilidad (float p) { probabilidad = p;}
// .... otras declaraciones....
        private float probabidad;
```

}

Modifique y/o genere el código necesario si a usted se le pide lanzar una excepción cuando alguien intente asignar una probabilidad superior a 1.0.

```
public class myclass{
      public void setProbabilidad (float p)throws ProbabilidadException {
          if (p > 1.0) {
            throw(new ProbabilidadException(p,"Probabilidad Mayor que
uno"));
          }

          probabilidad = p;
      }

// .... otras declaraciones....

      private float probabilidad;
}

class ProbabilidadException extends Exception
{
    public String msg;  // to make it simple, I have declared them
public.
    public float p;

    ProbabilidadException(float _p, String s)
    {
      p=_p;
      s=msg;
    }
}
```

5.-

a) ¿Bajo qué circunstancias usted está obligado a crear una thread (o hilo) implementando la interfaz Runnable?

Cuando no podemos heredar de Thread por estar ya heredando de otra clase. Java no permite múltiple herencia pero si podemos implementar múltiples interfaces y al mismo tiempo derivar de otra clase.

b) Sea h1 y h2 dos hilos, ¿Cómo podemos asegurar que h1 pase a estado dead (muerto o terminado) con posterioridad a la muerte o término de h2?

En h1 debemos tener una referencia a h2, supongamos rh2.
Luego antes de retornar o salir del método run() de h1 debemos poner:
rh2.join();

c) Se tiene el siguinete segmento:

```
...
        synchronized void A()   { /* parte A */.......}
        synchronized void B()   { ./* parte B */......}
        void C()  {
                System.out.println("Feliz navidad un tanto adelantada ....");
                synchronized (this) {
                ...../* parte c2  */
                }
```

```
        }
....
```

Indique qué partes del código pueden ser ejecutadas en forma concurrente por múltiples threads.

En estos métodos se hace referencia directa o indirectamente al mismo monitor. Por ello sólo una thread podrá estar ejecutando las instrucciones de alguno de los segmentos parte A, parte B, o parte C. Sí se puede concurremcia múltiple entre un hilo en alguna de las partes ya indicadas y otros en la sentencia de salida System.out.println("Feliz navidad un taanto adelantada...");

Table 9.1 Summary of list operations

Constructors and Assignment		
`list<T> v;`	Default constructor	$O(1)$
`list<T> v (aList);`	Copy constructor	$O(n)$
`l = aList`	Assignment	$O(n)$
`l.swap (aList)`	Swap values with another list	$O(1)$
Element Access		
`l.front ()`	First element in list	$O(1)$
`l.back ()`	Last element in list	$O(1)$
Insertion and Removal		
`l.push_front (value)`	Add value to front of list	$O(1)$
`l.push_back (value)`	Add value to end of list	$O(1)$
`l.insert (iterator, value)`	Insert value at specified location	$O(1)$
`l.pop_front ()`	Remove value from front of list	$O(1)$
`l.pop_back ()`	Remove value from end of list	$O(1)$
`l.erase (iterator)`	Remove referenced element	$O(1)$
`l.erase (iterator,iterator)`	Remove range of elements	$O(1)$ [a]
`l.remove (value)`	Remove all occurrences of value	$O(n)$
`l.remove_if (predicate)`	Removal all values that match condition	$O(n)$
Size		
`l.empty ()`	True if collection is empty	$O(1)$
`l.size ()`	Return number of elements in collection	$O(n)$ [b]
Iterators		
`list<T>::iterator itr`	Declare a new iterator	$O(1)$
`l.begin ()`	Starting iterator	$O(1)$
`l.end ()`	Ending iterator	$O(1)$
`l.rbegin ()`	Starting backwards moving iterator	$O(1)$
`l.rend ()`	Ending backwards moving iterator	$O(1)$
Miscellaneous		
`l.reverse ()`	Reverse order of elements	$O(n)$
`l.sort ()`	Place elements into ascending order	$O(n \log n)$
`l.sort (comparison)`	Order using comparison function	$O(n \log n)$
`l.merge (list)`	Merge with another ordered list	$O(n)$

a. Freeing the memory used by erased cells will require time proportional to the number of elements deleted.

b. Some implementations keep track of the number of elements in a list, and thus can determine the size in $O(1)$.