Primer Certamen 1er. Sem 2004

Todas las preguntas tienen igual puntaje.

1.

a) Usando los niveles de madurez definidos en CMM, ¿qué nivel le corresponde a una empresa que se caracteriza por comprometer productos de software cuya calidad depende de las personas que allí trabajan? Justifique en no más de **tres líneas**.

Nivel 1 o Inicial. La carencia de un proceso bien definido y sin mayor control hace que los resultados sean tan buenos o malos como el equipo de personas que lo desarrollan.

b) Explique por qué ya hay esfuerzo dedicado a pruebas en la fase de Elaboración. Muestre un bloque del V-Model que correspondería a esta actividad.

Junto a la definición de los elementos centrales de la arquitectura es preciso planear y eventualmente desarrollar los sistemas de prueba a usar para verificar el buen funcionamiento de módulos de software al ser integrados entre si. En V-Model es el bloque de "Integration and Test Planning"

NOTA: Si argumentamos por el lado de desarrollo basado en ciclos cortos, se explica que en cada fase haya pruebas, en particular en la de Elaboración. En este caso todos los módulos de "testing planning" deberían ser mencionados.

c) Explique por qué la ejecución de programas Java es más lenta que programas escritos en C. ¿Qué significa la compilación justo a tiempo (Just-in-time compilation)?

La compilación de Java genera código intermedio (Bytecode) el cual es luego interpretado por la máquina virtual Java mientras que en C la compilación genera código binario nativo el cual directamente ejecutado por la plataforma donde se corre. La interpretación de byetcode a binario nativo hace que Java sea más lento.
La compilación justo a tiempo se refiere a la mantener el código nativo de aquellas partes ya interpretadas del bytecode de modo que cada sentencia sea interpretada sólo una vez. Por ejemplo es un lazo for el cuerpo es interpretado una vez y luego el código generado es ejecutado directamente las iteraciones siguientes.

d) Mencione dos diferencias y dos semejanzas entre clases abstractas e interfaces.

Diferencias:

i) Las clases abstractas son usadas como clases bases sobre las cuales otras son derivadas. La interfaces no son clases bases.

ii) Una clase sólo puede mantener relación es un con una clase abstracta, pero puede mantener relación es un con muchas interfaces.

iii) Una clase abstracta puede definir una implementación para alguno de sus métodos, una Interfaz no tiene implementación alguna.

Semejanzas:

i) No es posible crear instancias directas de una clase abstracta ni de una Interfaz.

ii) Instancias de una clase que implementa una interfaz o que hereda de una clase abstracta pueden ser usadas en donde se espera un objeto que cumpla con la interfaz o del tipo de la clase abstracta.

iii) En ambos casos tenemos una relación del tipo es-un entre la clase base y la abstracta y entre la clase que implementa y su interfaz.

e) Alguien dice: "En la actualidad las applets requieren que los servidores web tengan soporte Java, al menos el jre." ¿Es verdadera esta afirmación? Explique.

Falso. Sólo el navegador debe tener soporte para Java. En la actualidad esto se logra con el uso de plugins los cuales ocupan la jre presente en la máquina del navegador.

2. Programe la clase Stack la cual permite en Java instancear objetos similares al conseguido con el siguiente código en C:

```
int datastack[100];
int datatop = 0;
void init() {
      datatop=0;
}
void push (int val) {
      if (datato <100)
            datastack [datatop++] = val;
}
void top () {
      if (datatop >0 )
            return (datastack [datatop-1]);
}
int pop() {
      if (datatop >0)
            return (datastack [--datatop]);
      return 0;
}
```

```
class Stack
{
 Stack () {
  datatop=0;
  datastack = new int[CAPACIDAD];
 }
 public void push(int val) {
  if (datatop < CAPACIDAD)
   datastack[datatop++]=val;
 }
 public int top () {
  if (datatop >0)
   return(datastack[datatop-1]);
 }
 public int pop() {
  if (datatop > 0)
   return(datastack[--datatop]);
```

```
  return 0;
 }
 public void reset(){
  datatop=0;
 }
 private int[] datastack;
 private datatop;
 private final int CAPACIDAD=100;
}
```

3. Tenemos:

```
class Employee { ..... }
class Manager extends Employee { .... }
Employee e = new Manager(...);
Manager m = (Manager)e;
```

Supongamos que deseamos invocar todos los métodos definidos en Employee y Manager primero sobre ***m*** y luego sobre ***e***. Indique bajo qué condiciones cada invocación es permitida y qué implementación serán ejecutada. Considere distintos casos (algunos métodos sobremontados, otros no, etc..)

La invocación sobre *m* será permitida sólo si el método está definido como público en Employee o en Manager.

La invocación sobre e será permitida sólo si el método está definido como público en Employee.

En ambos casos el código ejecutado corresponde a la implentación en Employee de aquellos métodos no sobremontados, la implementación de Manager de aquellos sobremontados, y en el caso de *m* además la implementación en Manager de aquellos métodos sólo de Manager.

4. Modifique el programa mimic.java visto en clases para que el texto ingresado actualice el título de la ventana.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class Mimic extends JFrame {
  MimicGUI gui = new MimicGUI(this);
  public Mimic() {
    setTitle("Mimic");
    setSize( 250, 100);

    // fast way to setup closeable window listener object
    addWindowListener( new WindowAdapter() {
```

```
          public void windowClosing( WindowEvent e) {
            System. exit( 0);
          }
        }
    );
    getContentPane().add(gui);
    setVisible(true);
  }

  public static void main( String[] args) {
    Mimic mimic = new Mimic();
   // new Mimic();

  }
}

class MimicGUI extends JPanel {
  private JLabel label = new JLabel(" Echo appears here");
  private JTextField quote = new JTextField( 20);
  private MimicListener listener = new MimicListener( this);
  private JFrame frame;

  public MimicGUI(JFrame f) {
    frame=f;
    // add quote and label to window
    add(quote);
    add(label);
    // register listener with quote object
    quote.addActionListener(listener);
  }

  public void updateLabel() {
    label.setText(quote.getText());
    frame.setTitle(quote.getText());
  }
}

class MimicListener implements ActionListener {
  private MimicGUI gui;
  public MimicListener( MimicGUI guiref) {
    gui = guiref;
  }
  // method required by action listener interface
  public void actionPerformed( ActionEvent e) {
    gui.updateLabel();
  }
```

```
}
```

5. a) Haga un Applet que muestre cuatro números enteros: cada uno representa el número de veces que se ha invocado los métodos init(), start(), stop(), y destroy() respectivamente.
   b) Indique el contenido del archivo html que permita mostrar el applet.

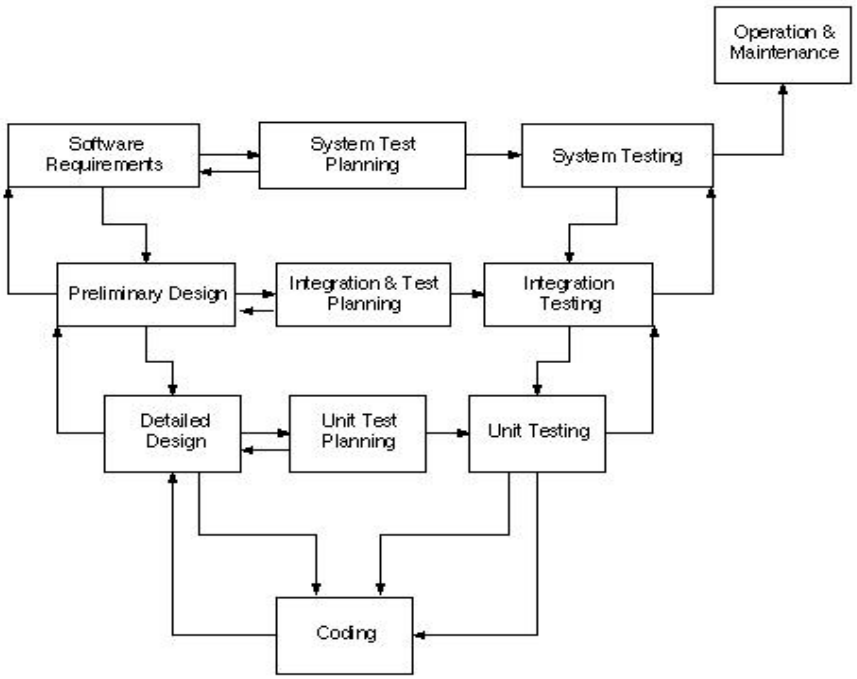
a)

```
import java.awt.*;
import java.applet.*;
import javax.swing.*;

public class Contadores extends JApplet {
 public void init(){
  cinit=cstart=cstop=cdestroy=0;
  cinit++;
  init = new JLabel("init= "+cinit);
  start = new JLabel("start= "+cstart);
  stop = new JLabel("stop= "+cstop);
  destroy = new JLabel("destroy= "+cdestroy);
  Container contentPane = getContentPane();
  contentPane.setLayout(new FlowLayout());
  contentPane.add(init);
  contentPane.add(start);
  contentPane.add(stop);
  contentPane.add(destroy);
 }

 public void start(){
  cstart++;
  start.setText("start= "+cstart);
 }

 public void stop() {
  cstop++;
  stop.setText("stop= "+cstop);
 }

 public void destroy(){
  cdestroy++;
  destroy.setText("destroy= "+cdestroy);
 }
 private JLabel init, start,stop,destroy;
 private int cinit, cstart, cstop, cdestroy;
}
```

b)

```
<applet code="Contadores.class" width="300" height="200">
</applet>
```

V-MODEL



"Mimic.java"

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class Mimic extends JFrame {
  MimicGUI gui = new MimicGUI();
  public Mimic() {
    setTitle("Mimic");
    setSize( 250, 100);

    // fast way to setup closeable window listener object
    addWindowListener( new WindowAdapter() {
      public void windowClosing( WindowEvent e) {
        System. exit( 0);
      }
      }
                  );
    getContentPane().add(gui);
    setVisible(true);
  }

  public static void main( String[] args) {
    Mimic mimic = new Mimic();
```

```
   // new Mimic();

  }
}

class MimicGUI extends JPanel {
  private JLabel label = new JLabel(" Echo appears here");
  private JTextField quote = new JTextField( 20);
  private MimicListener listener = new MimicListener( this);

  public MimicGUI() {
    // add quote and label to window
    add(quote);
    add(label);
    // register listener with quote object
    quote.addActionListener(listener);
  }

  public void updateLabel() {
    label.setText(quote.getText());
  }
}

class MimicListener implements ActionListener {
  private MimicGUI gui;
  public MimicListener( MimicGUI guiref) {
    gui = guiref;
  }
  // method required by action listener interface
  public void actionPerformed( ActionEvent e) {
    gui.updateLabel();
  }
}
```