Prueba Especial, Programación II

Prof. Rodrigo Olivares
Ayud. Juan Carlos Rojas
Septiembre 07, 2016

Instrucciones:

- El puntaje máximo de la prueba especial es 100%, siendo el 60% el mínimo requerido para aprobar.
- Responda cada pregunta en el lugar indicado. No se aceptarán recorrecciones de pruebas respondidas con lápiz grafito.
- El tiempo máximo de la evaluación es de 90 minutos.
- La prueba especial es **individual**. Cualquier intento de copia, será sancionado con nota **1,0**.

1. *40pts.* De las siguentes afirmaciones, encierre en un círculo la o las alternativas correctas (*3pts c/u*).

i. La orientación a objeto es:
(a) Un paradigma de programación procedural.
(b) Un paradigma de programación estucturado.
(c) Una herramienta de programación.
(d) Un lenguaje de programación.
(e) Ninguna de las anteriores.

ii. El principio de ocultamiento:
(a) Es una técnica que protege el estado de una entidad.
(b) Es útil en enfoques procedurales.
(c) En Java, se logra con los modificadores de acceso.
(d) Es encapsular el conocimiento de una entidad.
(e) Ninguna de las anteriores.

iii. Una interface:
(a) Tiene al menos un método implementado.
(b) Tiene todos sus métodos abstractos.
(c) Es factible de ser implementada.
(d) Es factible de ser extendida.
(e) Ninguna de las anteriores.

vi. La herencia múltiple:
(a) Permite heredar diverso compartimiento.
(b) Apoya el principio ocultamiento.
(c) Apoya el principio de encapsulamiento.
(d) En Java se desarrolla implementado clases abstractas.
(e) En Java se desarrolla implementado interfaces.

v. Un thread:
(a) Es un flujo de un proceso en memoria.
(b) Es un proceso que se ejecuta en memoria.
(c) Puede ser creado como clase en Java.
(d) Puede ser instanciado como atributo.
(e) Ninguna de las anteriores.

vi. Para una hebra o hilo se debe:
(a) Iniciar con el método run.
(b) Iniciar con el método start.
(c) Sobreescribir el método run.
(d) Sobreescribir el método start.
(e) Dormir (sleep) la hebra.

vii. En el ciclo de vida de una hebra, el estado:
(a) New crea la hebra.
(b) Runnable ejecuta siempre la hebra.
(c) Blocked se ejecuta, sin importar estados internos.
(d) Dead es invocado generalmente por el método sleep.
(e) Yield, verifica el desempeño del estado Runnable.

viii. Los bloqueos de recursos compartidos se consiguen:
(a) Package, bloqueando los accesos a las clases internas.
(b) Clase, bloqueando métodos y atributos de la clase.
(c) Atributo, declarándolos como static.
(d) Objeto, declarando los métodos como synchronized.
(e) Ninguna de las anteriores

ix. Referente a JFrame:
(a) Habitualmente se usa para crear la ventana principal.
(b) getPaneContent() obtiene el panel principal.
(c) setAdd() permite agregar componentes al panel.
(d) setSize() permite dimensionar la ventana.
(e) Ninguna de las anteriores

x. Para realizar acciones desde un botón Se requiere:
(a) Crear una clase que implemente un ActionEvent.
(b) Crear una clase que implemente un ActionListener.
(c) Re-escribir el método actionEvent(ActionPerformed).
(d) Re-escribir el método actionPerformed(ActionEvent).
(e) Agregar la instancia de la clase oyente, al botón.

2. *70pts.* Desarrolle una aplicación en JAVA, con interfaz de usuario que permita mostrar en algún componente de salida, la información bancaria de abono y giro de un cliente, simulado por procesos concurrentes. Para ellos se solicita lo siguiente:

   (a) Construir interfaz de usuario en JAVA que contenga como mínimo lo que se muestra en la Figura 1). Los botones "Detener" permiten pausar/dormir el proceso en cuestión, cambiando su etiqueta a "Procesar". El botón "Procesar" permite despertar/levantar el proceso en cuestión *(20 pts)*.

   (b) Construir los procesos concurrentes:

      i. ProcesoBancoAbono: Permite registrar abonos aleatorios (entre $1 y $10.000 CLP) sin límite máximo *(15 pts)*.

      ii. ProcesoBancoGiro: Permite registrar giros aleatorios (entre $1 y $10.000 CLP) sin límite mínimo *(15 pts)*.

   (c) Luego de construir los procesos concurrentes, debe desarrollar la clase de recurso compartido *(15 pts)*

   (d) Por último, desarrolle la clase principal de la aplicación *(5 pts)*.

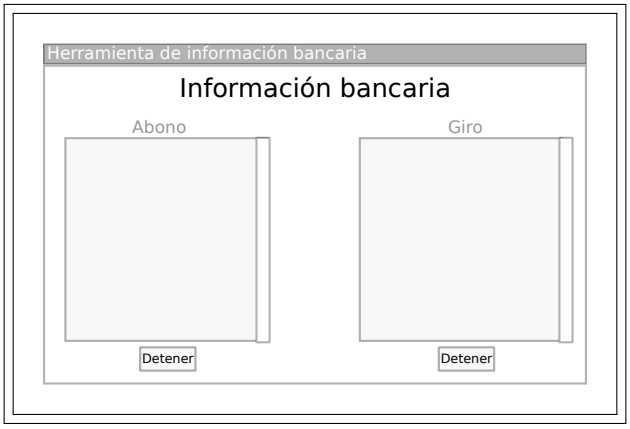

Figura 1: Interfaz de usuario

Interfaz de usuario y método principal

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class CuentaCorrienteView extends JFrame {
  private JTextArea textoAbono, textoGiro;
  private JButton accionAbono, accionGiro;
  private JPanel panelTitulo, panelAbono, panelGiro;
  private JScrollPane scrollPaneAbono, scrollPaneGiro;
  private final String TEXT_DETENER = "Detener";
  private final String TEXT_PROCESAR = "Procesar";
  private CuentaCorriente cc;
  private ThreadCuentaCorrienteDepositar tccd;
  private ThreadCuentaCorrienteGirar tccg;
```

```
public CuentaCorrienteView() {
    super("Herramienta de informacin bancaria");
    initialComponent();
    startProccess();
}

private void initialComponent() {
    textoAbono = new JTextArea(10, 15);
    textoGiro = new JTextArea(10, 15);
    accionAbono = new JButton(TEXT_DETENER);
    accionGiro = new JButton(TEXT_DETENER);
    panelTitulo = new JPanel();
    panelAbono = new JPanel();
    panelGiro = new JPanel();
    scrollPaneAbono = new JScrollPane(textoAbono);
    scrollPaneGiro = new JScrollPane(textoGiro);
    setLayout(new BorderLayout());
    panelTitulo.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
    panelTitulo.add(new JLabel("Informacin bancaria"));
    add(panelTitulo, BorderLayout.NORTH);
    panelAbono.setLayout(new BorderLayout(10, 10));
    panelAbono.add(new JLabel("    Abono    "), BorderLayout.NORTH);
    panelAbono.add(scrollPaneAbono, BorderLayout.CENTER);
    accionAbono.addActionListener(new OyenteAbono());
    panelAbono.add(accionAbono, BorderLayout.SOUTH);
    add(panelAbono, BorderLayout.WEST);
    panelGiro.setLayout(new BorderLayout(10, 10));
    panelGiro.add(new JLabel("    Giro    "), BorderLayout.NORTH);
    panelGiro.add(scrollPaneGiro, BorderLayout.CENTER);
    accionGiro.addActionListener(new OyenteGiro());
    panelGiro.add(accionGiro, BorderLayout.SOUTH);
    add(panelGiro, BorderLayout.EAST);
    setSize(400, 400);
    setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
}

private void startProccess() {
    cc = new CuentaCorriente();
    tccd = new ThreadCuentaCorrienteDepositar(cc, this);
    tccg = new ThreadCuentaCorrienteGirar(cc, this);
    tccd.start();
    tccg.start();
}

public void setTextAbono(String text) {
    textoAbono.append(text + "\n");
}

public void setTextGiro(String text) {
    textoGiro.append(text + "\n");
}

class OyenteAbono implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e != null) {
            if (TEXT_DETENER.equals(e.getActionCommand())) {
              accionAbono.setText(TEXT_PROCESAR);
              tccd.suspend();
      } else {
          accionAbono.setText(TEXT_DETENER);
          tccd.resume();
      }
    }
  }
```

```
  }

  class OyenteGiro implements ActionListener {

      @Override
      public void actionPerformed(ActionEvent e) {
          if (e != null) {
              if (TEXT_DETENER.equals(e.getActionCommand())) {
            accionGiro.setText(TEXT_PROCESAR);
                tccg.suspend();
        } else {
            accionGiro.setText(TEXT_DETENER);
            tccg.resume();
        }
      }
    }
  }
}
```

Recurso compartido

```
public class CuentaCorriente {
    private int saldo = 0;
    public synchronized void depositar(int dinero, CuentaCorrienteView cuentaCorrienteView)
    {
        saldo += dinero;
        cuentaCorrienteView.setTextAbono("Deposito: " + dinero + " Saldo: " + getSaldo());
        notifyAll();
    }
    public synchronized void girar(int dinero, CuentaCorrienteView cuentaCorrienteView) {
        saldo -= dinero;
        cuentaCorrienteView.setTextGiro("Giro: " + dinero + " Saldo: " + getSaldo());
        notifyAll();
    }
    private synchronized int getSaldo() {
        return saldo;
  }
}
```

Proceso Depósito

```
public class ThreadCuentaCorrienteDepositar extends Thread {

    private final CuentaCorriente cuentaCorriente;
    private final CuentaCorrienteView cuentaCorrienteView;

    public ThreadCuentaCorrienteDepositar(CuentaCorriente cuentaCorriente,
    CuentaCorrienteView cuentaCorrienteView){
        this.cuentaCorriente = cuentaCorriente;
        this.cuentaCorrienteView = cuentaCorrienteView;
    }

    @Override
    public void run() {
        while (true) {
            cuentaCorriente.depositar((int) (10000 * Math.random()), cuentaCorrienteView);
        }
    }
}
```

Proceso Giro

```
public class ThreadCuentaCorrienteGirar extends Thread {

    private final CuentaCorriente cuentaCorriente;
```

```
    private final CuentaCorrienteView cuentaCorrienteView;

    public ThreadCuentaCorrienteGirar(CuentaCorriente cuentaCorriente, CuentaCorrienteView
    cuentaCorrienteView) {
        this.cuentaCorriente = cuentaCorriente;
        this.cuentaCorrienteView = cuentaCorrienteView;
    }

    @Override
    public void run() {
        while (true) {
            cuentaCorriente.girar((int) (10000 * Math.random()), cuentaCorrienteView);
        }
    }
}
```

Principal

```
public class Main {
    public static void main(String[] args) {
        CuentaCorrienteView view = new CuentaCorrienteView();
    }
}
```