Programación orientada a objetos. Segundo parcial 2002.

Viernes 21 de junio de 2002.

1. Supongan que existen dos clases diferentes que no tienen métodos con signaturas en común. ¿Es posible tener un objeto capaz de ejecutar todos los métodos de esas clases? ¿Cómo? ¿Hay más de una manera? ¿Cuáles? No pueden escribir los métodos; deben usar los que ya están implementados.

   Además de contestar las preguntas anteriores programen las respuestas en Java usando las siguientes clases:

```
public class Cliente {
  public int getNumeroCliente() {…}
  public void setNumeroCliente(int numeroCliente) {…}
}

public class Alumno {
  public String getEMail() {…}
  public void setEMail(String eMail) {…}
}
```

   10 puntos

2. Sean las siguiente clases:

```
public class Rectangle {
  public void paint() {…}
}

public class Circle {
  public void draw() {…}
}
```

   Los métodos `Rectangle.paint()` y `Circle.draw()` son semánticamente iguales pero tienen nombres diferentes porque han sido programados en distinto momento y por diferentes personas.

   Un fragmento del programa que usa estas clases aparece a continuación:

```
Set s = new HashSet();

s.add(new Rectangle());
/* Otros rectángulos son creados y agregados aquí… */

s.add(new Circle());
/* Otros círculos son creados y agregados aquí… */

Iterator i = s.iterator();
while (i.hasNext()) {
  Object o = i.next();
  if (o instanceof Rectangle) {
    (Rectangle)o.paint();
  }
  else if (o instanceof Circle) {
    (Circle)o.draw();
  }
}
```

   Ahora su jefe le pone a usted a cargo de este programa y le pregunta si está escrito usando buenas prácticas de programación orientada a objetos; en caso de que no sea así le pedirá que lo escriba de nuevo –el jefe no sabe programación orientada a objetos y confía en su respuesta o se hace el que no sabe para evaluar si usted sabe-. ¿Qué le contestaría a su jefe? Si fuera necesario escribir el programa de nuevo ¿cómo lo escribiría y porqué? ¿Hay más de una manera de hacerlo? ¿Cuáles? ¿Cuál elegiría y porqué?.

10 puntos

```
Stack {
  Vector();
  int top = -1;

  empty() {
    (elements.isEmpty());
  }

  Object peek() {
     (elements.get(top));
  }

  Object () {
    Object result = elements.remove(top);
    top--;
     (result);
  }

   push(Object item) {
    elements.add(item);
    top++;
  }
}
```

3. Sean las clases e interfaces siguientes:

```
public class BankAccount {
  double balance() {…}
}

public interface Account {
  double getBalance();
}

public interface Persist {
  void storeOn(OutputStream s);
}

public class ObjectFiler {
  void store(Object o, OutputStream s) {…}
}
```

Terminen de programar las modificaciones necesarias en la clase BankAccount que ahora tiene la siguiente declaración:

```
public class BankAccount implements Account, Persist {
}
```

10 puntos

4. En el obligatorio que hizo con su grupo: ¿Existía un plan de pagos como una colección de cuotas o de pagos? Justifique la respuesta. ¿Los objetos de qué clase realizaban los cálculos de las cuotas para un préstamo amortizable? ¿Qué opinión le merece un obligatorio en el que además de clases para préstamos y planes de pago hay una clase para obtener el plan de pagos de un préstamo y otra para calcular cuotas y moras?

30 puntos

5. Defina **composición** y **delegación**. Enumere ventajas y desventajas de programar la reutilización de código usando composición y delegación en lugar de herencia.

15 puntos –5 la definición y 10 la enumeración-

6. ¿Qué relación hay entre **composición** y **delegación** e **interfaces** en Java?

5 puntos

7. ¿Qué es el principio de sustitución? ¿Qué relación tiene con los términos **tipo** y **herencia por restricción**? ¿Y con **polimorfismo**?

15 puntos –3 la definición y 4 cada relación -

8. ¿Puede haber polimorfismo en un lenguaje en un lenguaje en el que no hay herencia? Justifiquen su respuesta.

5 puntos