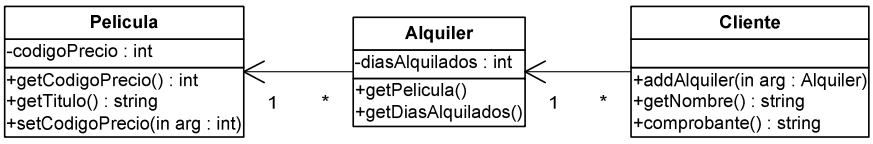
Programación orientada a objetos. Segundo parcial 2004.

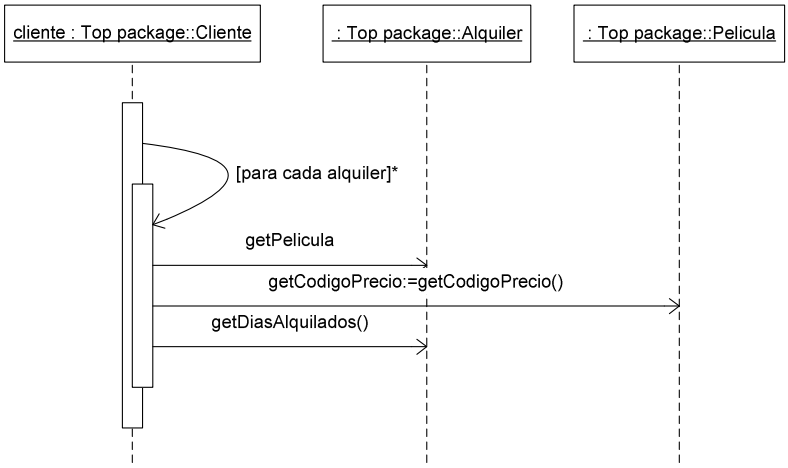
Viernes 18 de junio de 2004.

Pongan el nombre sólo en la primera hoja. Contesten las preguntas en orden en que están formuladas agrupando las respuestas de las preguntas de la 1 a la 5 y de la 6 a la 9 porque serán corregidas por diferentes profesores. Escriban sólo del anverso de la hoja y eviten doblarlas. Gracias y mucha suerte.

```
/* Pelicula.java */
public class Pelicula {
  public static final int INFANTIL = 2;
  public static final int REGULAR = 0;
  public static final int ESTRENO = 1;

  public String m_titulo;
  private int m_codigoPrecio;

  public Pelicula(String titulo, int codigoPrecio) {
       m_titulo = titulo;
       m_codigoPrecio = codigoPrecio;
  }

  public int getCodigoPrecio() {
       return m_codigoPrecio;
  }

  public String getTitulo() {
       return m_titulo;
  }

  public void setCodigoPrecio(int arg) {
       m_codigoPrecio = arg;
  }
}

/* Alquiler.java */
public class Alquiler {
  private Pelicula m_pelicula;
  private int m_diasAlquilados;

  public Alquiler(Pelicula pelicula, int diasAlquilados) {
    m_pelicula = pelicula;
    m_diasAlquilados = diasAlquilados;
  }

  public int getDiasAlquilados() {
    return m_diasAlquilados;
  }

  public Pelicula getPelicula() {
    return m_pelicula;
  }
}

/* Cliente.java */
import java.util.*;

public class Cliente {
  private String m_nombre;
  private Vector m_alquileres = new Vector();

  public Cliente(String nombre) {
    m_nombre = nombre;
  }
```

```
  public void addAlquiler(Alquiler arg) {
    m_alquileres.addElement(arg);
  }

  public String getNombre() {
    return m_nombre;
  }

  public String comprobante() {
    double importeTotal = 0;
    int puntosAlquilerFrecuente = 0;
    Iterator alquileres = m_alquileres.iterator();
    String result = "Registro de alquiler " + getNombre() + "\n";
    while (alquileres.hasNext()) {
      double importe = 0;
      Alquiler alquiler = (Alquiler)alquileres.next();

      // Determinar importe de cada línea
      switch(alquiler.getPelicula().getCodigoPrecio()) {
        case Pelicula.REGULAR:
          importe =+ 2;
          if (alquiler.getDiasAlquilados() > 2)
            importe += (alquiler.getDiasAlquilados() - 2) * 1.5;
            break;
        case Pelicula.ESTRENO:
          importe += alquiler.getDiasAlquilados() * 3;
          break;
        case Pelicula.INFANTIL:
          importe += 1.5;
          if (alquiler.getDiasAlquilados() > 3)
            importe += (alquiler.getDiasAlquilados() - 3) * 1.5;
          break;
      }

      // Agregar puntos de alquiler frecuente
      puntosAlquilerFrecuente++;

      // Agregar boons por alquiler de estrenos de dos días
      if ((alquiler.getPelicula().getCodigoPrecio() ==
        Pelicula.ESTRENO) && alquiler.getDiasAlquilados() > 1)
        puntosAlquilerFrecuente++;

      // Mostrar importes de este alquiler
      result += "\t" + alquiler.getPelicula().getTitulo() + "\t" +
        String.valueOf(importe) + "\n";
      importeTotal += importe;
    }
    // Agregar líneas al pie
    result += "Total es " + String.valueOf(importeTotal) + "\n";
    result += "Usted ganó " + String.valueOf(puntosAlquilerFrecuente) +
      " puntos de alquiler frecuente";
    return result;
  }
}
```





Este programa es muy simple. Es un programa que calcula e imprime un comprobante para un cliente de un videoclub. Al programa se le indica las películas que alquila un cliente y por cuanto tiempo. El programa calcula el importe dependiendo del tiempo por el que es alquilada la película e identifica el tipo de película. Hay tres tipos de película: regulares, de niños y estrenos. Además de calcular el importe, el comprobante también calcula los puntos de alquiler frecuente, que varían dependiendo de si la película es un estreno.

El programa es correcto; cumple con los requisitos. Compila y funciona.

1. Critique este programa justificando cada una de sus críticas. Indique cómo resolvería cada problema encontrado. Escriba nuevamente el programa.

   Pista: Concéntrese en reducir el método `comprobante()` de la clase `Cliente`. Aplique los conceptos de encapsulación, modularidad y jerarquía.

   30 puntos

2. Sean dos clases diferentes que no tienen métodos con signaturas en común. ¿Es posible tener un objeto capaz de recibir mensajes con selectores para todos los métodos de esas clases? ¿Cómo? ¿Hay más de una manera? ¿Cuáles? No pueden escribir los métodos nuevamente; deben usar los que ya están implementados.

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

```
public class Impresora {
  public void imprimir(String texto) {…}
}

public class Fax {
  public String recibirFax() {…}
  public void enviarFax(String fax) {…}
}
```

   10 puntos

3. Sean las clases e interfaces siguientes:

```
public class LinkedList {
  void add(Object o) {…}
  Object remove(int index) {…}
}

public interface Stack {
  void push(Object o);
  Object pop();
}
```

```
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 `LinkedList` que ahora tiene la siguiente declaración:

```
public class LinkedList implements Stack, Persist {
}
```

10 puntos

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

   10 puntos

5. Imagine que tiene tres clases `Rol`, `Recurso`, `Operación`. Pueden existir un número indeterminado de roles, recursos y operaciones distintas. Proponga una forma de agregar/quitar permisos para realizar cierta operación sobre cierto recurso para un rol dado, de forma que el rol pueda responder si tiene permiso o no para una operación sobre un cierto recurso. Programe la solución en Java.

   10 puntos

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

   10 puntos

7. Expliquen en forma clara y precisa cómo funciona el **encadenamiento estático** y el **encadenamiento dinámico**. ¿Es posible tener ambos tipos de encadenamiento en lenguajes en los que no se declara el tipo de las variables y argumentos?

   Muestren las diferencias entre encadenamiento estático y dinámico, con un pequeño programa en Java usando las siguientes clases, considerando en un caso que Java sólo soporta encadenamiento estático y en el otro solo dinámico.

   ```
   public class Persona {
     public void caminar() {…}
   }

   public class Cliente extends Persona {
     public void caminar() {…}
   }

   public class Estudiante extends Persona {
     public void caminar() {…}
   }
   ```

   10 puntos

8. Erich Gamma, autor del libro Design Patterns, afirma que "…la composición es una alternativa a la herencia; nueva funcionalidad se obtiene ensamblando o componiendo objetos para obtener funcionalidad más compleja…". El autor afirma que, en el contexto de la reutilización, toda implementación que use herencia se puede cambiar por una equivalente que use composición y delegación. ¿Qué ventajas puede tener la composición y delegación sobre la herencia?

   10 puntos