Programación orientada a objetos. Examen julio 2003.

Viernes 20 de junio de 2003.

```
import java.util.*;

public class Discoteca {

  static Map discos = new HashMap();

  public static void main(String[] args) {
    Discoteca.addCD("Abraxas", "Carlos Santana");
    Discoteca.addCancion("Abraxas", "Samba pa ti");
    Discoteca.addCancion("Abraxas", "Oye como va");

    Discoteca.addCD("The wall", "Pink Floyd");
    Discoteca.addCancion("The wall", "In The Flesh?");
    Discoteca.addCancion("The wall", "The Thin Ice");
    Discoteca.addCancion("The wall", "Another Brick In The Wall, Part 1");
    Discoteca.addCancion("The wall", "Another Brick In The Wall, Part 2");
    Discoteca.addCancion("The wall", "Another Brick In The Wall, Part 3");

    Discoteca.addCD("Grandes éxitos", "Grupo Casino");
    Discoteca.addCancion("Grandes éxitos", "Azuquita' pa'l café");
    Discoteca.addCancion("Grandes éxitos", "Caballo 'e la sabana");

    Discoteca.addDVD("Pulp Fiction", "Acción");
    Discoteca.addActor("Pulp Fiction", "John Travolta");
    Discoteca.addActor("Pulp Fiction", "Uma Thurman");

    Discoteca.addDVD("Grease", "Romántica");
    Discoteca.addActor("Grease", "John Travolta");
    Discoteca.addActor("Grease", "Olivia Newton John");

    Discoteca.addDVD("The Matrix", "Ciencia Ficción");
    Discoteca.addActor("The Matrix", "Keanu Reaves");

    System.out.println(Discoteca.favoritos());
  }

  static Collection favoritos() {
    Collection result = new HashSet();
    Set keys = discos.keySet();
    Iterator i = keys.iterator();
    while (i.hasNext()) {
      String disco = (String)i.next();
      List contenido = (List)discos.get(disco);
      if (contenido.get(0).equals("cd")) {
        String interprete = (String)contenido.get(1);
        if (interprete.equals("Carlos Santana") ||
          interprete.equals("Pink Floyd")) {
          result.add(disco);
        }
      }
      if (contenido.get(0).equals("dvd")) {
        Set actores = (Set)contenido.get(1);
        String genero = (String)contenido.get(2);
        if ((actores.contains("John Travolta") ||
          actores.contains("Uma Thurman")) &&
          (genero.equals("Acción") || genero.equals("Ciencia Ficción"))) {
          result.add(disco);
        }
      }
    }
    return result;
  }

  static void addCD(String cd, String interprete) {
    List contenido = new ArrayList();
```

```
    contenido.add(0, "cd"); /* id */
    contenido.add(1, interprete); /* intérprete */
    contenido.add(2, new HashSet()); /* canciones */
    discos.put(cd, contenido);
  }

  static void addCancion(String cd, String cancion) {
    List contenido = (List)discos.get(cd);
    Set canciones = (Set)contenido.get(2);
    canciones.add(cancion);
  }

  static void addDVD(String dvd, String genero) {
    List contenido = new ArrayList();
    contenido.add(0, "dvd"); /* id */
    contenido.add(1, new HashSet()); /* actores */
    contenido.add(2, genero); /* género */
    discos.put(dvd, contenido);
  }

  static void addActor(String dvd, String actor) {
    List contenido = (List)discos.get(dvd);
    Set actores = (Set)contenido.get(1);
    actores.add(actor);
  }
}
```

Las clases anteriores implementan una pequeña discoteca que permite guardar discos compactos con canciones y discos de vídeo digital con películas. La discoteca guarda, para cada disco compacto, el nombre del disco compacto, el nombre del intérprete y la lista de canciones; y para cada disco de video digital, el nombre de la película y la lista de actores. El programa principal imprime en la consola los discos compactos y los discos de vídeo digital favoritos. Un disco compacto es favorito cuando su intérprete es Carlos Santana o Pink Floyd y un disco de vídeo digital es favorito cuando actúan en la película John Travolta o Uma Thurman en una película de acción o de ciencia ficción.

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

1. Critique este programa desde el punto de vista de la aplicación de los conceptos de programación orientada a objetos vistos en clase. Indique claramente que conceptos no se aplican o se aplican en forma incorrecta. Además de enumerar los conceptos, defínalos brevemente y justifique su crítica.

   20 puntos.

2. Indique cómo resolvería cada problema encontrado. Justifique su respuesta.

   10 puntos.

3. Escriba nuevamente el programa. ¿Cambiaría la implementación si se agregara a la clase Discoteca los métodos `Collection getDVDsActor(String actor)` y `Collection getCDsInterprete(String interprete)`? El primer método retorna una colección con los discos de video digital de un actor y el segundo una colección con los discos compactos de un intérprete. Justifique su respuesta y agregue los métodos.

   20 puntos.

4. 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 nuevamente 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

5. Expliquen en forma clara y precisa cómo funciona el **encadenamiento estático** y el **encadenamiento dinámico**.

   Muestren las diferencias entre encadenamiento estático y dinámico con un pequeño fragmento de código en Java usando el siguiente ejemplo, suponiendo sólo por un instante que Java soportara encadenamiento estático.

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


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

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

10 puntos

6. ¿Puede una **precondición**, **poscondición** o **invariante de clase** hacer referencia a **atributos privados** de la clase?

   ¿Puede una **invariante** o una **poscondición** hacer referencia a los **argumentos de un método**?

   Justifiquen su respuesta.

   En caso que alguna respuesta sea afirmativa, programen un pequeño ejemplo en Java para esa o esas respuestas.

10 puntos

7. 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.

   ¿La afirmación contraria es cierta? Es decir, ¿toda solución que use **composición y delegación** se puede implementar también usando **herencia**? Justifiquen la respuesta y den un ejemplo en Java.

10 puntos

8. Un objeto puede tener más de un **tipo** y varios objetos de diferentes clases pueden tener el mismo **tipo**. Muestren con un pequeño fragmento de código de Java un ejemplo de un objeto con más de un tipo y otro con varios objetos de diferentes clases y el mismo tipo.

10 puntos