Programación orientada a objetos. Examen diciembre 2004.

Miércoles 24 de noviembre de 2004.

```
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. Critiquen este programa desde el punto de vista de la aplicación de los conceptos de programación orientada a objetos vistos en clase, indicando al menos dos de los conceptos más importantes que no se aplican o se aplican en forma incorrecta. Definan esos dos conceptos y justifiquen su crítica.

   20 puntos.

2. 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.

```
/* Perro.java */
public class Perro {
  public static int PERRO = 1;
  public static int PERRA = 2;
  private String m_nombre;
  private int m_sexo;

  public Perro(…) {…}
  public String getNombre() {…}
  public void setNombre(…) {…}
  public int getSexo() {…}
  public Cruza cruzar(…) {…}
}
```

```
/* Cruza.java */
import java.util.*;

public class Cruza {
  private Perro m_perro;
  private Perro m_perra;
  private Collection m_prole = new ArrayList();
  public Cruza(Perro perro, Perro perra) {…}
  public void addCachorro(Perro cachorro) {
    m_prole.add(cachorro);
  }
  public Iterator getProle() {
    return m_prole.iterator();
  }
}
```

3. Sean las clases `Perro` y `Cruza` para registrar las cruzas entre perros de raza. Completen los métodos que dicen "…". Tengan en cuenta que:
   - Un perro no puede cambiar de raza.
   - Un perro no puede cambiar de sexo.
   - No se puede cambiar el perro y la perra de una cruza.
   - No se puede quitar cachorros de una cruza.

   10 puntos

4. Definan en forma breve y precisa **precondición** y **poscondición**. Agreguen a las clases `Perro` y `Cruza` del ejercicio anterior las precondiciones y poscondiciones que consideren necesarias para asegurar que:
   - Un perro no se puede cruzar consigo mismo.
   - Un perro se debe cruzar con otro de su misma raza.
   - Un pero no se puede cruzar con otro del mismo sexo.
   - Un perro no se puede cruzar con sus padres ni con ninguno de sus hermanos.
   - Una cruza no puede cambiar los perros que intervinieron en ella.

   Agreguen estas afirmaciones al código del ejercicio anterior usando la cláusula **`assert`** de Java. No tienen porqué escribir nuevamente toda la clase, mientras indiquen claramente dónde se inserta cada modificación. Pueden agregar otros métodos además de los que ya existen.

   10 puntos

5. Programen un caso de prueba `TestPerro` en JUnit para la clase `Perro`.

   10 puntos

```
public class Animal {
  public void comer() {
  …
  };
}
```

6. Usando las clases `Perro` y `Animal` anteriores creen una nueva clase que tenga los métodos de ambas simultáneamente. Muestren todas las formas diferentes de hacerlo si hubiera. No pueden escribir los métodos; deben usar los que ya están implementados.

   Hagan un programa en Java en el que crean un objeto al que le envían el mensaje `comer()` y el mensaje `cruzar()`.

   10 puntos

```
public class Carnivoro extends Animal {
  public void comer() {
  …
  };
}
```

7. Expliquen en forma clara y precisa cómo funciona el **encadenamiento estático** y el **encadenamiento dinámico** en esos lenguajes. ¿El concepto es aplicable a los 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 clases provistas, suponiendo sólo por un instante que Java soportara encadenamiento estático.

   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 programa en Java, usando las clases provistas, un ejemplo de un objeto con más de un tipo y otro con varios objetos de diferentes clases y el mismo tipo.

   10 puntos