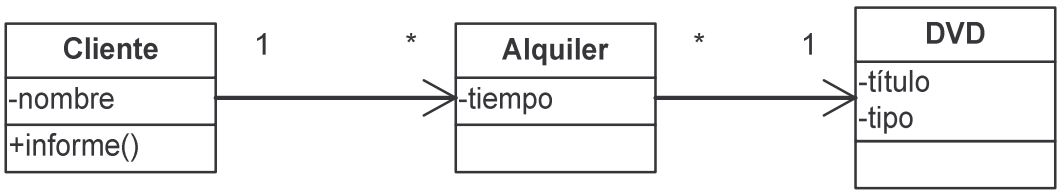
Caso práctico
Alquiler de películas en un vídeo-club

Adaptado de "*Refactoring*" © Martin Fowler, 2000

Supongamos que tenemos que desarrollar una aplicación que gestione los alquileres de DVDs en un vídeo-club.

Inicialmente, nuestro diagrama de clases sería similar al siguiente:




```
public class DVD
{
  // Constantes simbólicas

  public static final int INFANTIL = 2;
  public static final int NORMAL   = 0;
  public static final int NOVEDAD  = 1;

  // Variables de instancia

  private String _titulo;
  private int    _tipo;

  // Constructor

  public DVD (String titulo, int tipo)
  {
    _titulo = titulo;
    _tipo   = tipo;
  }
```

```
  // Acceso a las variables de instancia

  public int getTipo()
  {
    return _tipo;
  }

  public void setTipo (int tipo)
  {
    _tipo = tipo;
  }

  public String getTitulo ()
  {
    return _titulo;
  }
}
```

```
public class Alquiler
{
  private DVD _dvd;
  private int _tiempo;

  public Alquiler (DVD dvd, int tiempo)
  {
    _dvd = dvd;
    _tiempo = tiempo;
  }

  public int getTiempo()
  {
    return _tiempo;
  }

  public DVD getDVD()
  {
    return _dvd;
  }
}
```

```
import java.util.Vector;


public class Cliente
{
  // Variables de instancia

  private String _nombre;
  private Vector _alquileres =new Vector();

  // Constructor

  public Cliente (String nombre)
  {
    _nombre = nombre;
  }

  // Acceso a las variables de instancia

  public String getNombre()
  {
    return _nombre;
  }

  // Registrar alquiler

  public void nuevoAlquiler (Alquiler alquiler)
  {
    _alquileres.add(alquiler);
  }

  // Emitir un informe del cliente

  public String informe()
  {
    double      total;
    double      importe;
    int         puntos;
    int         i;
    Alquiler    alquiler;
    String      salida;

    total  = 0;
    puntos = 0;
    salida = "Informe para " + getNombre() + "\n";
```

```
// Recorrido del vector de alquileres

for (i=0; i<_alquileres.size(); i++) {

    importe  = 0;
    alquiler = (Alquiler) _alquileres.get(i);

    // Importe del alquiler

    switch (alquiler.getDVD().getTipo()) {

      case DVD.NORMAL:
           importe += 2;
           if (alquiler.getTiempo()>2)
              importe += (alquiler.getTiempo()-2)*1.5;
           break;

      case DVD.NOVEDAD:
           importe += alquiler.getTiempo() * 3;
           break;

      case DVD.INFANTIL:
           importe += 1.5;
           if (alquiler.getTiempo()>3)
              importe += (alquiler.getTiempo()-3)*1.5;
           break;
    }

    // Programa de puntos

    puntos++;

    if (  (alquiler.getDVD().getTipo()==DVD.NOVEDAD)
       && (alquiler.getTiempo()>1) )
       puntos++;                      // Bonificación

    // Mostrar detalles del alquiler

    salida += "\t" + alquiler.getDVD().getTitulo()
            + "\t" + String.valueOf(importe) + " €\n";

    // Acumular total

    total += importe;
}
```

```
    // Pie del informe
    salida += "IMPORTE TOTAL = "
            + String.valueOf(total) + " €\n";
    salida += "Dispone de "
            + String.valueOf(puntos) + " puntos\n";
    return salida;
  }
}
```

Paso 1: Extraer método de **`informe()`**

El método `informe` es excesivamente largo…

```
public class Cliente…

  public double precio (Alquiler alquiler)
  {
    double importe = 0;
    switch (alquiler.getDVD().getTipo()) {
      case DVD.NORMAL:
           importe += 2;
           if (alquiler.getTiempo()>2)
              importe += (alquiler.getTiempo()-2)*1.5;
           break;
      case DVD.NOVEDAD:
           importe += alquiler.getTiempo() * 3;
           break;
      case DVD.INFANTIL:
           importe += 1.5;
           if (alquiler.getTiempo()>3)
              importe += (alquiler.getTiempo()-3)*1.5;
           break;
    }
    return importe;
  }
```

```
public String informe()
{
  double      total;
  double      importe;
  int         puntos;
  int         i;
  Alquiler    alquiler;
  String      salida;

  total  = 0;
  puntos = 0;
  salida = "Informe para " + getNombre() +  "\n";

  for (i=0; i<_alquileres.size(); i++) {

      alquiler = (Alquiler) _alquileres.get(i);
      importe  = precio(alquiler);

      // Programa de puntos

      puntos++;

      if ((alquiler.getDVD().getTipo()==DVD.NOVEDAD)
         && (alquiler.getTiempo()>1))
         puntos++;                    // Bonificación

      // Mostrar detalles del alquiler

      salida += "\t" + alquiler.getDVD().getTitulo()
              + "\t" + String.valueOf(importe) + "€\n";

      // Acumular total

      total += importe;
  }

  // Pie del recibo

  salida += "IMPORTE TOTAL = "
          + String.valueOf(total) + " €\n";

  salida += "Dispone de "
          + String.valueOf(puntos) + " puntos\n";

  return salida;
}
```

Debemos comprobar que calculamos correctamente los precios, para lo que preparamos una batería de casos de prueba:

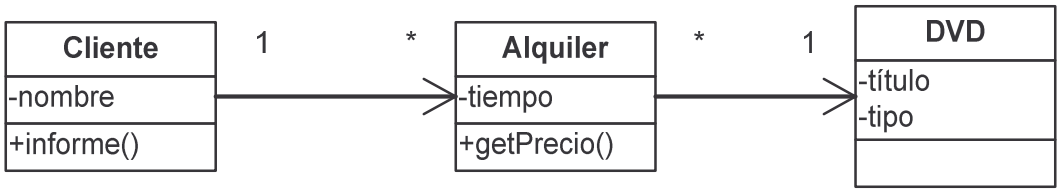
```
import junit.framework.*;

public class AlquilerTest extends TestCase
{
  private Cliente  cliente;
  private DVD      casablanca;
  private DVD      indy;
  private DVD      shrek;
  private Alquiler alquiler;
                                                  // Infraestructura
  public static void main(String args[])
  {
    junit.swingui.TestRunner.main (
          new String[] {"AlquilerTest"});
  }

  public AlquilerTest(String name)
  {
    super(name);
  }

  public void setUp ()
  {
    cliente = new Cliente("Kane");
    casablanca = new DVD("Casablanca", DVD.NORMAL);
    indy = new DVD("Indiana Jones XIII", DVD.NOVEDAD);
    shrek = new DVD("Shrek", DVD.INFANTIL);
  }
                                                  // Casos de prueba
  public void testNormal1 ()
  {
    alquiler = new Alquiler(casablanca,1);
    assertEquals( cliente.precio(alquiler), 2.0, 0.001);
  }

  public void testNormal2 ()
  {
    alquiler = new Alquiler(casablanca,2);
    assertEquals( cliente.precio(alquiler), 2.0, 0.001);
  }

  public void testNormal3 ()
  {
    alquiler = new Alquiler(casablanca,3);
    assertEquals( cliente.precio(alquiler), 3.5, 0.001);
  }
```

```
  public void testNormal7 ()
  {
    alquiler = new Alquiler(casablanca,7);
    assertEquals( cliente.precio(alquiler), 9.5, 0.001);
  }
  public void testNovedad1 ()
  {
    alquiler = new Alquiler(indy,1);
    assertEquals( cliente.precio(alquiler), 3.0, 0.001);
  }
  public void testNovedad2 ()
  {
    alquiler = new Alquiler(indy,2);
    assertEquals( cliente.precio(alquiler), 6.0, 0.001);
  }
  public void testNovedad3 ()
  {
    alquiler = new Alquiler(indy,3);
    assertEquals( cliente.precio(alquiler), 9.0, 0.001);
  }
  public void testInfantil1 ()
  {
    alquiler = new Alquiler(shrek,1);
    assertEquals( cliente.precio(alquiler), 1.5, 0.001);
  }
  public void testInfantil3 ()
  {
    alquiler = new Alquiler(shrek,3);
    assertEquals( cliente.precio(alquiler), 1.5, 0.001);
  }
  public void testInfantil4 ()
  {
    alquiler = new Alquiler(shrek,4);
    assertEquals( cliente.precio(alquiler), 3.0, 0.001);
  }
  public void testInfantil7 ()
  {
    alquiler = new Alquiler(shrek,7);
    assertEquals( cliente.precio(alquiler), 7.5, 0.001);
  }
}
```

11/11

Paso 2: Mover el método `precio()`

En realidad, `precio` no usa datos de `Cliente`, por lo que resulta más que razonable convertirlo en un método de la clase `Alquiler`…




```
public class Alquiler
{
  …
  public double getPrecio ()
  {
    double importe = 0;

    switch (getDVD().getTipo()) {

      case DVD.NORMAL:
           importe += 2;
           if (getTiempo()>2)
              importe += (getTiempo()-2) * 1.5;
           break;

      case DVD.NOVEDAD:
           importe += getTiempo() * 3;
           break;

      case DVD.INFANTIL:
           importe += 1.5;
           if (getTiempo()>3)
              importe += (getTiempo()-3) * 1.5;
           break;
    }

    return importe;
  }
}
```

- Cuando un método de una clase (`Cliente`) accede continuamente a los miembros de otra clase (`Alquiler`) pero no a los de su clase (`Cliente`), es conveniente mover el método a la clase cuyos datos utiliza. Además, el código resultante será más sencillo.

Como hemos cambiado `precio()` de sitio, tenemos que cambiar las llamadas a `precio()` que había en nuestra clase `Cliente`:

```
public class Cliente
{
  …

  public String informe()
  {
    …
    for (i=0; i<_alquileres.size(); i++) {
        alquiler = (Alquiler) _alquileres.get(i);
        importe  = alquiler.getPrecio();
        …
    }
    …
  }
}
```

Además, deberemos actualizar nuestros casos de prueba:

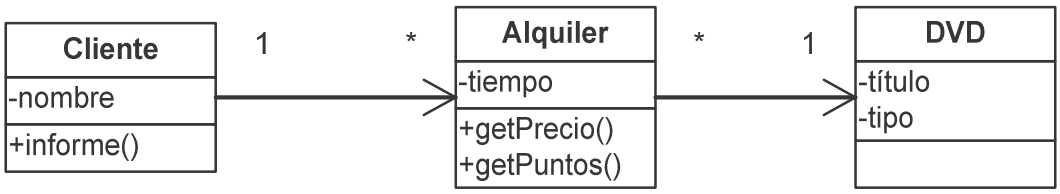
p.ej.

```
  public void testNormal3 ()
  {
    alquiler = new Alquiler(casablanca,3);
    assertEquals( alquiler.getPrecio(), 3.5, 0.001);
  }
```

Cuando hayamos realizado todos los cambios,
volveremos a ejecutar los casos de prueba
para comprobar que todo sigue funcionando correctamente.

11/11

Paso 3: Extraer el cálculo correspondiente al programa de puntos



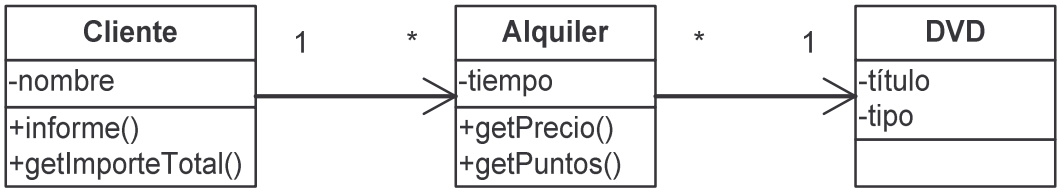

```
public class Alquiler
{
  …
  public int getPuntos ()
  {
    int puntos = 1;

    // Bonificación

    if (  (getDVD().getTipo() == DVD.NOVEDAD)
       && (getTiempo()>1))
       puntos++;

    return puntos;
  }
}
```

```
public class Cliente
{
  …
  public String informe()
  {
    …
    for (i=0; i<_alquileres.size(); i++) {
        alquiler = (Alquiler) _alquileres.get(i);
        importe  = alquiler.getPrecio();
        puntos  += alquiler.getPuntos();
        …
    }
    …
  }
}
```

Paso 4: Separar los cálculos de las operaciones de E/S

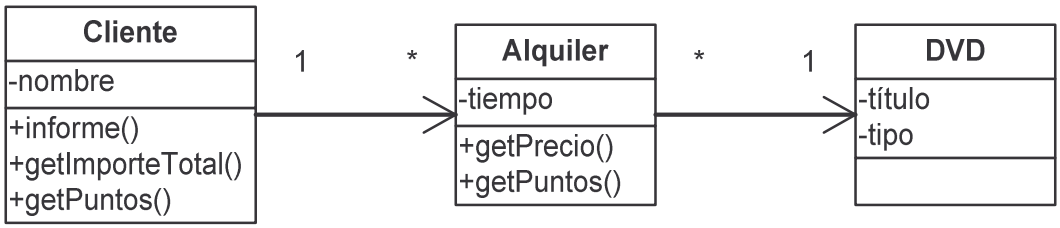
En el método `informe()`, estamos mezclando cálculos útiles con las llamadas a `System.out.println()` que generar el informe:

- Creamos un método independiente para calcular el gasto total realizado por un cliente:




```
public class Cliente…

  public double getImporteTotal ()
  {
    int      i;
    double   total;
    Alquiler alquiler;

    total = 0;

    for (i=0; i<_alquileres.size(); i++) {
        alquiler = (Alquiler) _alquileres.get(i);
        total   += alquiler.getPrecio();
    }

    return total;
  }
```

Creamos otro método independiente para calcular los puntos acumulados por un cliente en el programa de puntos del vídeo-club:




```
public class Cliente…

  public int getPuntos()
  {
    int      i;
    int      puntos;
    Alquiler alquiler;

    puntos = 0;

    for (i=0; i<_alquileres.size(); i++) {
        alquiler = (Alquiler) _alquileres.get(i);
        puntos  += alquiler.getPuntos();
    }

    return puntos;
  }
```

Al separar los cálculos de las operaciones de E/S, podemos preparar casos de prueba que comprueben el funcionamiento correcto del cálculo del total y del programa de puntos del vídeo-club.

- EJERCICIO -

- Tras los cambios anteriores en la clase `Cliente`, la generación del informe es bastante más sencilla que antes:

```
public class Cliente…

  public String informe()
  {
    int      i;
    Alquiler alquiler;
    String   salida;

    salida = "Informe para " + getNombre() + "\n";

    for (i=0; i<_alquileres.size(); i++) {

        alquiler = (Alquiler) _alquileres.get(i);

        salida += "\t"
                + alquiler.getDVD().getTitulo()
                + "\t"
                + String.valueOf(alquiler.getPrecio())
                + " €\n";
    }

    salida += "IMPORTE TOTAL = "
            + String.valueOf(getImporteTotal())
            + " €\n";

    salida += "Dispone de "
            + String.valueOf(getPuntos())
            + " puntos\n";

    return salida;
  }
```

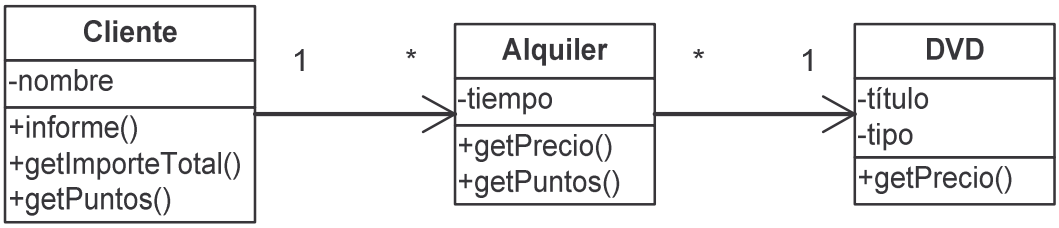
Paso 5: Nueva funcionalidad – Informes en HTML

Una vez que nuestro método `informe()` se encarga únicamente de realizar las tareas necesarias para generar el informe en sí, resulta casi trivial añadirle a nuestra aplicación la posibilidad de generar los informes en HTML (el formato utilizado para crear páginas web):

```
public class Cliente…

  public String informeHTML()
  {
    int         i;
    Alquiler    alquiler;
    String      salida;

    salida = "<H1>Informe para "
           + "<I>" + getNombre() + "</I>"
           + "</H1>\n";

    salida += "<UL>";

    for (i=0; i<_alquileres.size(); i++) {

        alquiler = (Alquiler) _alquileres.get(i);

        salida += "<LI>" + alquiler.getDVD().getTitulo()
                + "(" + alquiler.getPrecio() + " €)\n";
    }

    salida += "</UL>";

    salida += "<P>IMPORTE TOTAL = "
            + "<B>" + getImporteTotal() + " €</B>\n";


    salida += "<P>Dispone de "
            + "<I>" + getPuntos() + " puntos</I>\n";

    return salida;
  }
```

Paso 6: Mover el método `getPrecio()`

Movemos la implementación del método `getPrecio()` de la clase `Alquiler` al lugar que parece más natural si los precios van ligados a la película que se alquila:



public class DVD…

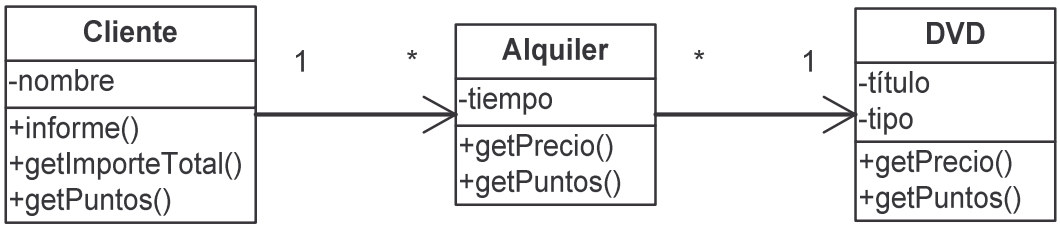
```
  public double getPrecio (int tiempo)
  {
    double importe = 0;

    switch (getTipo()) {

      case DVD.NORMAL:
           importe += 2;
           if (tiempo>2)
              importe += (tiempo-2) * 1.5;
           break;

      case DVD.NOVEDAD:
           importe += tiempo * 3;
           break;

      case DVD.INFANTIL:
           importe += 1.5;
           if (tiempo>3)
              importe += (tiempo-3) * 1.5;
           break;
    }

    return importe;
  }
```

public class Alquiler…

```
  public double getPrecio()
  {
    return _dvd.getPrecio(_tiempo);
  }
```

Paso 7: Mover el método `getPuntos ()`

Hacemos lo mismo con el método `getPuntos()`:




```
public class DVD…

  public int getPuntos (int tiempo)
  {
    int puntos = 1;

    // Bonificación

    if ((getTipo() == DVD.NOVEDAD) && (tiempo>1))
       puntos++;

    return puntos;
  }
```

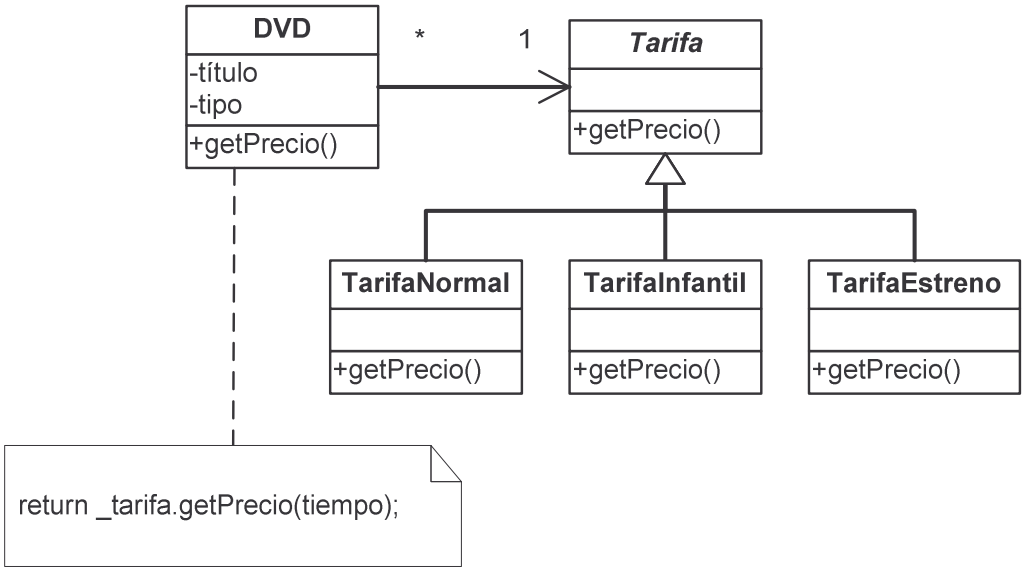
```
public class Alquiler…

  public int getPuntos ()
  {
    return _dvd.getPuntos(_tiempo);
  }
```

11/11

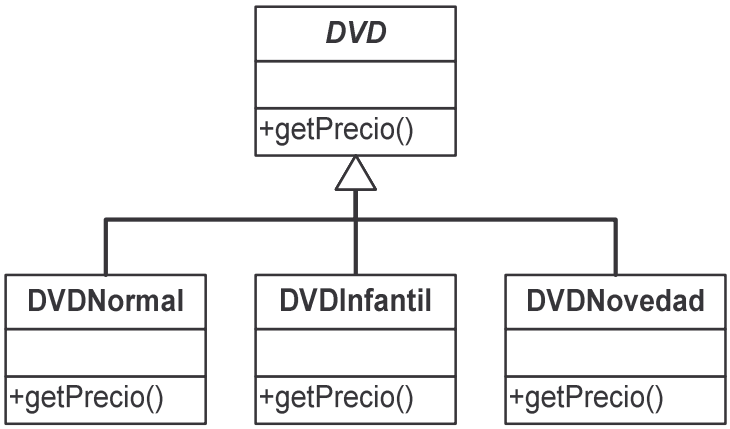
Como siempre, la ejecución de los casos de prueba nos confirma que todo funciona correctamente.

Paso 8: Reemplazar lógica condicional con polimorfismo

Queremos darle más flexibilidad a la forma en la que se fijan los precios de los alquileres de películas, para que se puedan añadir nuevas categorías (por ejemplo, en la creación de promociones) o para que se puedan ajustar las tarifas:



NOTA:
¿Por qué no creamos una jerarquía de tipos de películas?



Porque las películas pueden cambiar de categoría con el tiempo (dejan de ser una novedad) pero siguen manteniendo su identidad.

Nos creamos la jerarquía correspondiente a las políticas de precios:

```
public abstract class Tarifa
{
  public abstract int getTipo();
}
```

```
class TarifaNormal extends Tarifa
{
  public int getTipo()
  {
    return DVD.NORMAL;
  }
}
```

```
class TarifaInfantil extends Tarifa
{
  public int getTipo()
  {
    return DVD.INFANTIL;
  }
}
```

```
class TarifaEstreno extends Tarifa
{
  public int getTipo()
  {
    return DVD.NOVEDAD;
  }
}
```

A continuación, en la clase `DVD`, sustituimos el atributo `tipo` por un objeto de tipo `Tarifa`, en el cual recaerá luego la responsabilidad de establecer el precio correcto:

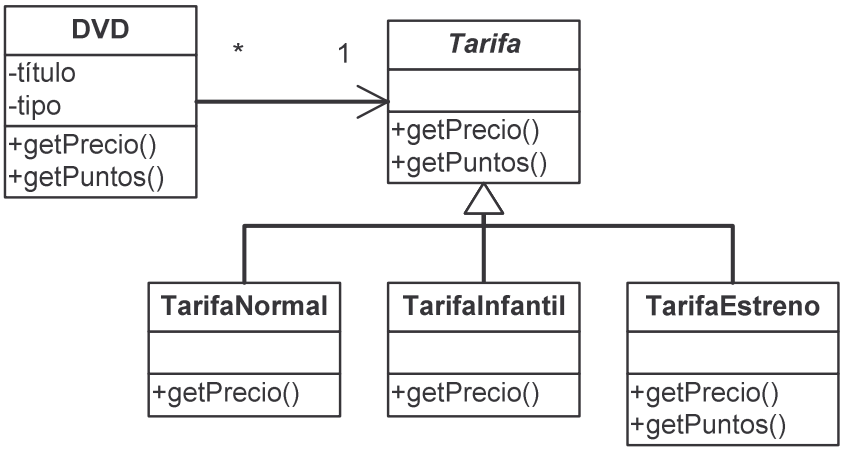
```
public class DVD…

  private String _titulo;
  private Tarifa _tarifa;


  public DVD (String titulo, int tipo)
  {
    _titulo = titulo;

    setTipo(tipo);
  }


  public int getTipo()
  {
    return _tarifa.getTipo();
  }


  public void setTipo (int tipo)
  {
    switch (tipo) {

      case DVD.NORMAL:
           _tarifa = new TarifaNormal();
           break;

      case DVD.NOVEDAD:
           _tarifa = new TarifaEstreno();
           break;

      case DVD.INFANTIL:
           _tarifa = new TarifaInfantil();
           break;
    }
  }
```

Finalmente, cambiamos la implementación de `getPrecio()`:

```
public abstract class Tarifa…
  public abstract double getPrecio (int tiempo);

class TarifaNormal extends Tarifa…
  public double getPrecio (int tiempo)
  {
    double importe = 2.0;
    if (tiempo>2)
       importe += (tiempo-2) * 1.5;
    return importe;
  }

class TarifaInfantil extends Tarifa
  public double getPrecio (int tiempo)
  {
    double importe = 1.5;
    if (tiempo>3)
       importe += (tiempo-3) * 1.5;
    return importe;
  }

class TarifaEstreno extends Tarifa
  public double getPrecio (int tiempo)
  {
    return tiempo * 3.0;
  }

public class DVD…
  public double getPrecio (int tiempo)
  {
    return _tarifa.getPrecio(tiempo);
  }
```

Paso 9: Reemplazar lógica condicional con polimorfismo II

Repetimos el mismo proceso de antes para el método `getPuntos()`:



public abstract class Tarifa…

```
  public int getPuntos (int tiempo)
  {
    return 1;
  }
```

class TarifaNovedad extends Tarifa…

```
  public int getPuntos (int tiempo)
  {
    return (tiempo>1)? 2: 1;
  }
```

public class DVD…

```
  public int getPuntos (int tiempo)
  {
    return _tarifa.getPuntos(tiempo);
  }
```

Paso 10: Mejoras adicionales

Todavía nos quedan algunas cosas que podríamos mejorar…

Las constantes simbólicas definidas en la clase `DVD` son, en realidad, meros identificadores que utilizamos para las diferenciar las distintas políticas de precios, por lo que deberíamos pasarlas a la clase genérica `Tarifa`.

Podríamos, directamente, eliminar esas constantes artificiales y forzar que, al crear un objeto de tipo `DVD`, el constructor reciba directamente como parámetro un objeto de alguno de los tipos derivados de `Tarifa` (lo que nos facilitaría el trabajo enormemente si se establecen nuevas políticas de precios).

Las distintas políticas de precios tienen características en común, por lo que podríamos reorganizar la jerarquía de clases teniendo en cuenta los rasgos comunes que se utilizan para establecer los precios (esto es, el precio base, el límite de tiempo y la penalización por tiempo).