Ejercicio 2 (ITIS)

La sucesión de **Padovan** es la secuencia de números enteros grandes P(n) definida por los siguientes valores iniciales

$$P(0) = P(1) = P(2) = 1,$$

y la siguiente relación de recurrencia

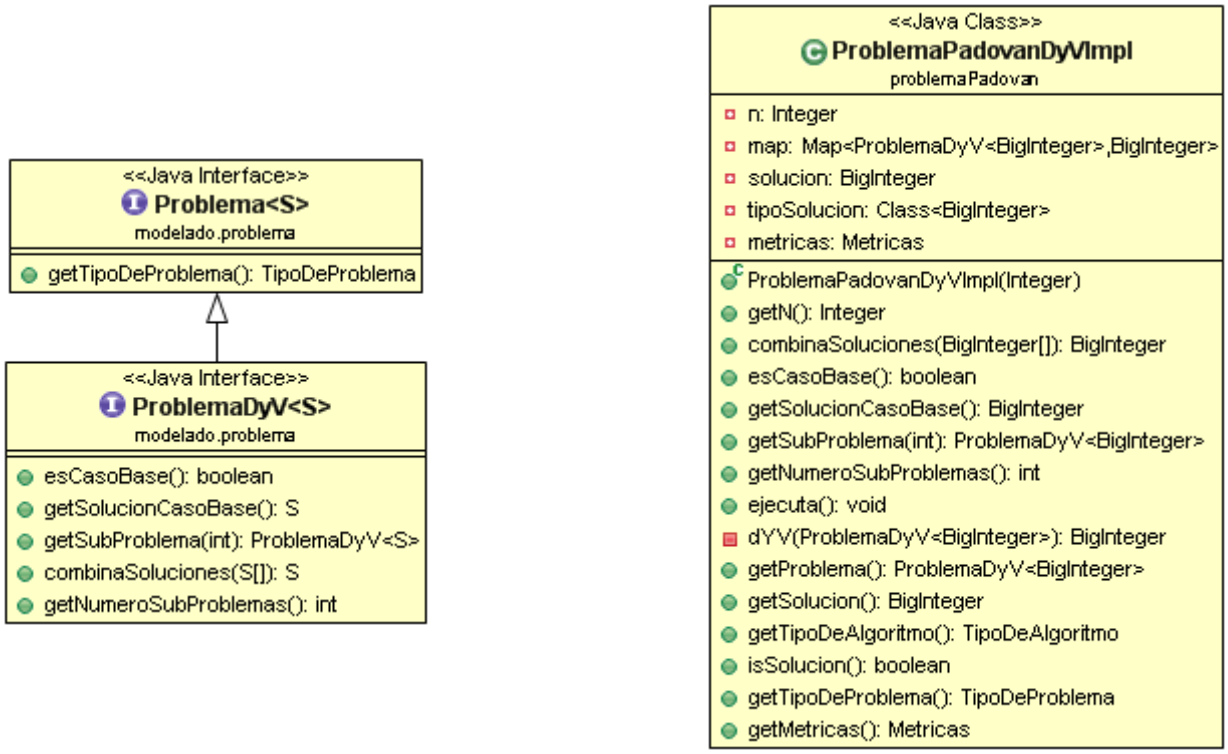
$$P(n) = P(n - 2) + P(n - 3)$$

Los primeros valores de P(n) son: 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37,...

Se pide:

a) Rellene la Ficha adjunta (que deberá entregar) siguiendo la información del problema de Padovan detallado anteriormente. Debe decidir cuál es la mejor opción para la técnica de Divide y Vencerás en este problema concreto: ¿con o sin memoria? Justifíquelo brevemente.

b) Implemente los métodos pertenecientes a la clase `ProblemaPadovanDyVImpl`: para ello, ayúdese de los diagramas UML proporcionados a continuación:



a. `private BigInteger dYV(ProblemaDyV<BigInteger> p)`. Resuelve la técnica Divide y Vencerás.
b. `public boolean esCasoBase()`. Indica si el problema que se está resolviendo es tan simple que no necesita recursión para su resolución.
c. `public BigInteger getSolucionCasoBase()`. Devuelve la solución al problema, resuelto de forma directa.
d. `public int getNumeroSubProblemas()`. Indica el número de subproblemas.
e. `public ProblemaDyV<BigInteger> getSubproblema(int i)`. Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo.
f. `public BigInteger combinaSoluciones (BigInteger[] soluciones)`. Combina cada solución de la lista de soluciones de manera que juntas resuelvan el problema completo.

Tiempo estimado: 45 min Puntuación: 4 puntos

Solución:

a) **(0,75 puntos)**

<table>
<tr><th colspan="2">Ficha</th></tr>
<tr><th colspan="2">Sucesión de Padovan</th></tr>
<tr><td colspan="2">Técnica: // Rellenar
Divide y Vencerás con Memoria

Breve justificación (con o sin memoria): // Rellenar
Al dividir el problema en subproblemas ((n-2) y (n-3)), aparecen repetidos los problemas más de una vez. Guardando en memoria las operaciones ya realizadas disminuye el tiempo computacional y el número de operaciones a realizar.</td></tr>
<tr><td colspan="2">Representación: Un objeto por problema</td></tr>
<tr><td>Tamaño: N</td><td></td></tr>
<tr><td>Propiedades Compartidas</td><td>// Rellenar
M, Memoria</td></tr>
<tr><td>Propiedades Individuales</td><td>// Rellenar
N, Integer no negativo</td></tr>
<tr><td colspan="2">Solución: BigInteger</td></tr>
<tr><td colspan="2">// Rellenar
$$P(n) = \begin{cases} M(n) & , si\ n \in M \\ 1 & , si\ n \leq 2 \\ P(n-2) + P(n-3) & , si\ n > 2 \end{cases}$$</td></tr>
<tr><td>Complejidad</td><td>// Rellenar
Si g(n) = k Θ(|I|)
// también es válido Θ(n)</td></tr>
</table>

b)

a. `private BigInteger dYV(ProblemaDyV<BigInteger> p)`. Resuelve la técnica Divide y Vencerás. **(0,75 puntos)**

```
private S dYV(ProblemaDyV<S> p) {
      S s;
      if (map.containsKey(p)) {
            s = map.get(p);
      } else if (p.esCasoBase()) {
            s = p.getSolucionCasoBase();
            map.put(p, s);
      } else {
            int numeroDeSubProblemas = p.getNumeroSubProblemas();
            S[] soluciones = Utiles.creaArray(numeroDeSubProblemas);
            for(int i = 0; i < numeroDeSubProblemas; i++){
                  ProblemaDyV<S> pr = p.getSubProblema(i);
                  s = dYV(pr);
                  soluciones[i]=s;
                  map.put(pr, s);
            }
            s = p.combinaSoluciones(soluciones);
            map.put(p, s);
      }
      return s;
}
```

b. `public boolean esCasoBase()`. Indica si el problema que se está resolviendo es tan simple que no necesita recursión para su resolución. **(0, 5 puntos)**

```
public boolean esCasoBase() {
            return getN() <= 2;
}
```

c. `public BigInteger getSolucionCasoBase()`. Devuelve la solución al problema, resuelto de forma directa. **(0,25 puntos)**

```
public BigInteger getSolucionCasoBase() {
            return BigInteger.ONE;
}
```

d. `public int getNumeroSubProblemas()`. Indica el número de subproblemas. **(0,25 puntos)**

```
public int getNumeroSubProblemas() {
            return 2;
}
```

e. `public ProblemaDyV<BigInteger> getSubproblema(int i)`. Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo. **(0,5 puntos)**

```
public ProblemaDyV<BigInteger> getSubProblema(int i) {
            return new ProblemaPadovanDyVImpl(getN()-i-2);
}
```

f. `public BigInteger combinaSoluciones (BigInteger[] soluciones)`. Combina cada solución de la lista de soluciones de manera que juntas resuelvan el problema completo. **(0,5 puntos)**

```
public BigInteger combinaSoluciones(BigInteger[] soluciones) {
            BigInteger sol = soluciones[0].add(soluciones[1]);
            return sol;
}
```