Ejercicio 1:

Un inversor está sopesando invertir en un activo financiero. Para poder estimar el riesgo que asume el inversor, nos piden que calculemos el *periodo de máximo riesgo* del activo. Se define el *periodo de máximo riesgo* como el periodo de tiempo donde el activo ha producido la pérdida máxima (medida en tanto por ciento). Para realizar estos cálculos nos proporcionan un array con los porcentajes de variaciones de rendimientos del activo en las últimas *N* sesiones. Las variaciones serán positivas si el valor del activo sube y negativas en caso contrario. Para el siguiente array:

-0.6003%	0.75%	-1.1%	0.23%	-0.4205%	0.3%	0.502%
sesión 0	sesión 1	sesión 2	sesión 3	sesión 4	sesión 5	sesión 6

El *periodo de máximo riesgo* se producirá entre la sesiones 2 y 4 y el porcentaje de variación que se produce en dicho periodo será de -1.1+0.23-0.4205 = -1.2905%. Este porcentaje de variación será el peor que podamos encontrar entre todos los posibles periodos entre dos sesiones.

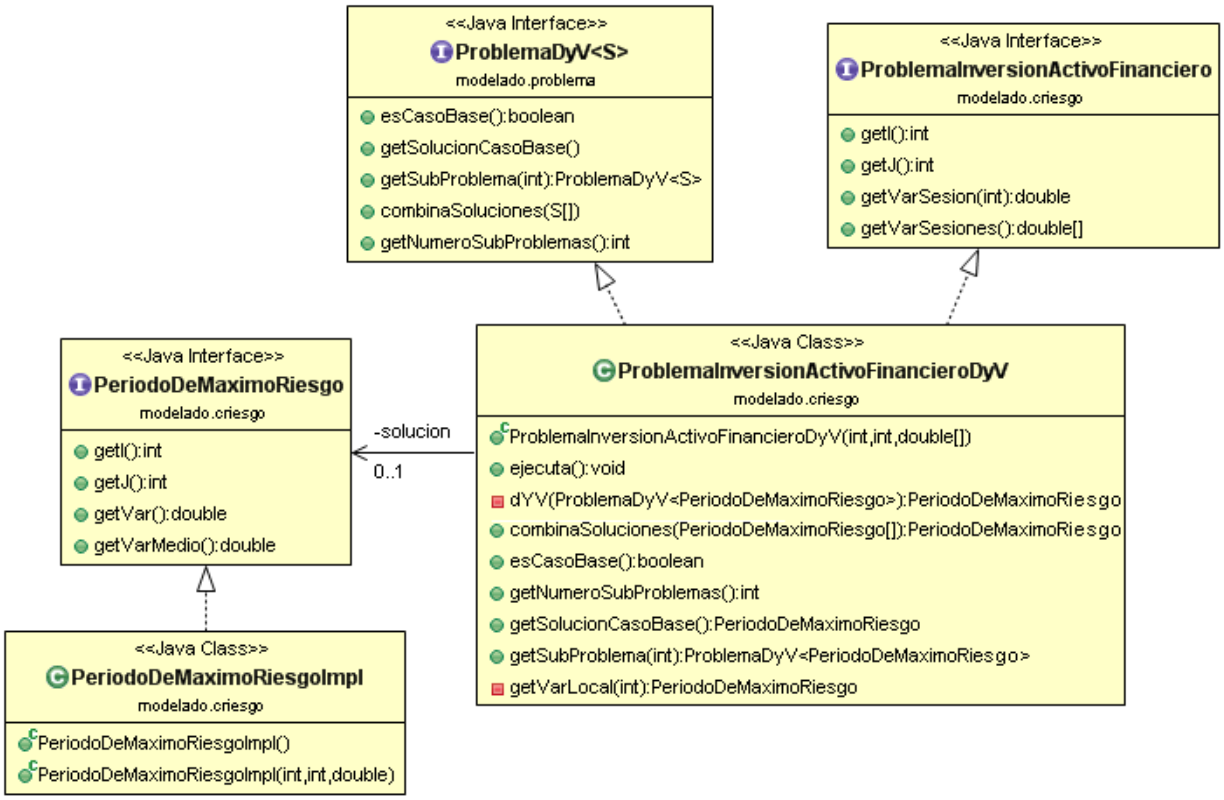
SE PIDE:

1) Indicar claramente si para resolver el problema descrito anteriormente debe utilizarse el esquema DyV con memoria o sin memoria.
2) Implementar el algoritmo dYV visto en clase, es decir, el método `private S dYV(ProblemaDyV<S> p)`.
3) Implementar los siguientes métodos de la clase `ProblemaInversionActivoFinancieroDyV`:
   a) `public boolean esCasoBase()`
   b) `public PeriodoDeMaximoRiesgo getSolucionCasoBase()`
   c) `public int getNumeroSubProblemas()`
   d) `public ProblemaDyV<PeriodoDeMaximoRiesgo> getSubProblema(int i)`
   e) `public PeriodoDeMaximoRiesgo combinaSoluciones (PeriodoDeMaximoRiesgo... soluciones)`
4) Determine el orden de complejidad del algoritmo diseñado:
   a) Defina el tamaño de la función
   b) Defina los casos mejor, peor y medio y calcule el orden del T(n) para cada caso.

NOTAS sobre el diagrama de clases:

- Los metodos `getI()`, `getJ()` y `getVar()` de la clase `PeriodoDeMaximoRiesgo` devuelven respectivamente la sesión inicial y final del periodo, y el porcentaje de variación de rendimiento del activo en dicho periodo.
- El método double `getVarSession(int pos)` de la clase `ProblemaInversionActivoFinanciero` devuelve la variación de rendimiento producida por el activo en la sesión `pos`. Y los métodos `getI()` y `getJ()` devuelven la primera y la última sesión del problema generalizado.
- Dispone del método `private PeriodoDeMaximoRiesgo getVarLocal(int s)` que calcula el periodo de máximo riesgo considerando que la sesión `s` siempre estará incluida en dicho periodo.

```
private PeriodoDeMaximoRiesgo getVarLocal(int s) {
   double sm = getVarSesion(s);
   int si = s, sj = s;
   double vs = sm;
   for(int k=s-1;getI()<=k;k--){
      vs = vs + getVarSesion(k);
      if (sm>vs) {
         sm = vs;
         si = k;
      }
   }
   vs = sm;
   for(int k=s+1;k<=getJ();k++){
     vs = vs + getVarSesion(k);
     if (sm>vs) {
        sm = vs;
        sj = k;
     }
   }
   return new PeriodoDeMaximoRiesgoImpl(si, sj, sm);
}
```



Puntuación: 33% **Tiempo estimado: 45 minutos**

SOLUCIÓN

1) DYV sin memoria

2) Esquema DYV sin memoria:

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

3) Métodos de la clase `ProblemaInversionActivoFinancieroDyV`:

```
public boolean esCasoBase() {
      return (getJ()-getI()+1==1 || getJ()<getI());
}

public PeriodoDeMaximoRiesgo getSolucionCasoBase() {
      PeriodoDeMaximoRiesgo rm = new PeriodoDeMaximoRiesgoImpl();
      if (getJ()-getI()+1==1)
        rm = new PeriodoDeMaximoRiesgoImpl(getI(), getJ(), getVarSesion(getI()));
      if (getJ()<getI())
        rm = new PeriodoDeMaximoRiesgoImpl(getI(), getJ(), Integer.MAX_VALUE);
      return rm;
}

public int getNumeroSubProblemas() {
      return 2;
}

public ProblemaDyV<PeriodoDeMaximoRiesgo> getSubProblema(int i) {
      int m = (getI() + getJ())/2;
      ProblemaDyV<PeriodoDeMaximoRiesgo> sp = null;
      if (i==0)
        sp = new ProblemaInversionActivoFinancieroDyV(getI(),m-1,getVarSesiones());
      if (i==1)
        sp = new ProblemaInversionActivoFinancieroDyV(m+1,getJ(),getVarSesiones());
      return sp;
}
```

```
public PeriodoDeMaximoRiesgo combinaSoluciones(PeriodoDeMaximoRiesgo... soluciones) {
    int s = (getI() + getJ())/2;
    PeriodoDeMaximoRiesgo rm = getVarLocal(s);
    PeriodoDeMaximoRiesgo rmIzq = soluciones[0];
    PeriodoDeMaximoRiesgo rmDer = soluciones[1];
    if (rm.getVar()>rmIzq.getVar()) {
        rm = rmIzq;
    }
    if (rm.getVar()>rmDer.getVar()) {
        rm = rmDer;
    }
    return rm;
}
```

4) Orden de complejidad

El tamaño será n = j – i
Los casos mejor, peor y medio son los mismos y el T(n) = 2 T(n/2) + O(n).
Aplicando el siguiente resultado de la resolución de ecuaciones en recurrencia:

$$T(n) = aT(n/b) + p(n)$$

Siendo $d = grado(p(n))$ su solución es

$$T(n) \in \begin{cases} \Theta(n^d) & \text{si } a < b^d \\ \Theta(n^d \log n) & \text{si } a = b^d \\ \Theta(n^{\log_b a}) & \text{si } a > b^d \end{cases}$$

entonces a=2, b=2 y d=1 y no encontraríamos en el segundo caso $a=b^d$ y el T(n) será de O(n log n) tanto para el caso mejor, peor y medio.