Ejercicio 2 II – ITIG (3 puntos):

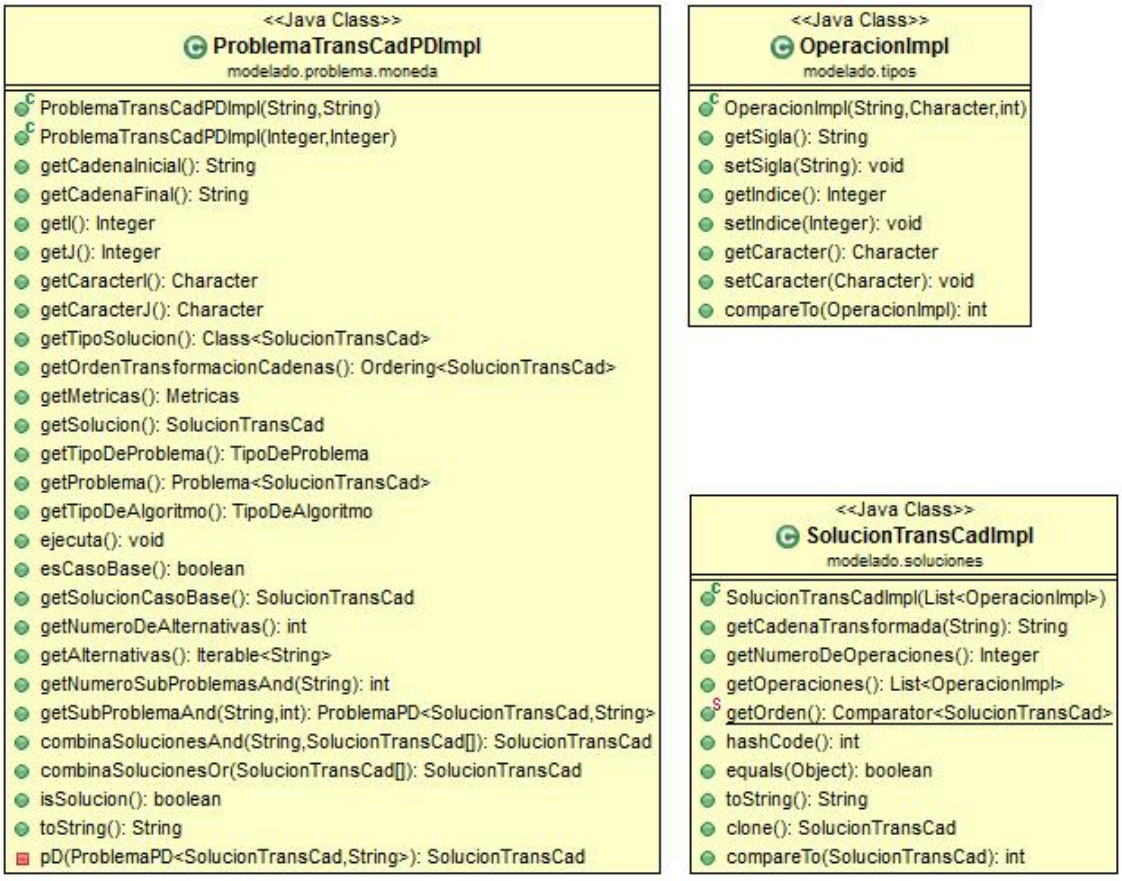
Sean *u* y *v* dos cadenas de caracteres. Se desea transformar *u* en *v* con las operaciones básicas del tipo siguiente:

- Eliminar un carácter.
- Añadir un carácter.
- Cambiar un carácter.

Por ejemplo, podemos pasar de *abbac* a *abcbc* en tres pasos:

- *abbac* → *abac* (eliminamos “b” en la posición 3).
- *abac* → *ababc* (añadimos “b” en la posición 4).
- *ababc* → *abcbc* (cambiamos “a” por “c” en la posición 3).

Sin embargo, se desea encontrar la mejor de todas las soluciones, es decir, aquella que realice la transformación en el menor número de operaciones, obteniéndose además la lista de operaciones. Para ello, se dispone del siguiente diagrama UML.



Se pide:

Implementar los siguientes métodos de la clase *ProblemaTransCadPDImpl* a partir de la ficha que se proporciona a continuación.

a) *boolean esCasoBase().*
b) *SolucionTransCad getSolucionCasoBase().*
c) *int getNumeroDeAlternativas().*
d) *int getNumeroSubproblemasAnd(String a).*
e) *ProblemaPD<SolucionTransCad, String> getSubproblemaAnd(String a, int i).*
f) *SolucionTransCad combinaSolucionesAnd(Integer a, SolucionTransCad… ls).*

Ficha	
Transformación de cadenas	
Técnica: Programación Dinámica	
Representación: Cada problema un conjunto de parámetros	
Tamaño: $N = I + J$	
Propiedades Compartidas	u, *cadena* v, *cadena*
Propiedades Individuales	I, *entero en* $[0, u.Size]$ J, *entero en* $[0, v.Size]$
Solución: SolucionTransCad	
Objetivo: Encontrar $s^{u,v,l,n}$ tal que n tenga el menor valor posible	
Alternativas: si $u_i = v_j$, entonces $A_{i,j} = \{N\}$; si $u_i \neq v_j$, entonces $A_{i,j} = \{E, A, C\}$	
$ptc(u, v) = tc(u.Size, v.Size)$ $tc(i,j) = \begin{cases} s^{u,v,\emptyset,0} & si\ i = 0, j = 0 \\ s^{u,v,\{i \text{ veces la operación } E\},i} & si\ i \neq 0, j = 0 \\ s^{u,v,\{j \text{ veces la operación } A\},j} & si\ i = 0, j \neq 0 \\ cO_{a \epsilon A_{i,j}}\big(cA(N, tc(i-1, j-1))\big) & si\ u_{i-1} = v_{j-1} \\ cO_{a \epsilon A_{i,j}}\big(cA(E, tc(i-1, j)), cA(A, tc(i, j-1)), cA(C, tc(i-1, j-1))\big) & si\ u_{i-1} \neq v_{j-1} \end{cases}$	
$cA(a_i, s)$*: Añade la operación* a_i *a la solución* s*, si* $a_i \neq N$	
$cO_{a \epsilon A_{i,j}}(s_{a_i}^{u,v,l,n})$*: Elige la solución con el valor menor de* n	
Complejidad	$\Theta(mn)$

Nota: la nomenclatura de las operaciones y el significado de la solución en la ficha se define como sigue:

- N → no hacer nada.
- E → eliminar un carácter.
- A → añadir un carácter.
- C → cambiar un carácter.
- $s^{u,v,l,n}$ → u y v son las cadenas, l es la lista de operaciones y n el número de operaciones.

Tiempo estimado: 60 minutos

Solución:

a. *boolean esCasoBase();*

```
public boolean esCasoBase() {
	return (getI() == 0) || (getJ() == 0);
}
```

b. *SolucionTransCad getSolucionCasoBase();*

```
public SolucionTransCad getSolucionCasoBase() {
	List<OperacionImpl> operaciones = Lists.newArrayList();

	if((getI() == 0) && (getJ() != 0)) {
		Character c = null;

		for(int i = 0; i < getJ(); i++) {
			c = getCadenaFinal().charAt(i);
			operaciones.add(new OperacionImpl("A", c, i));
		}
	}
	else if((getI() != 0) && (getJ() == 0)) {
		Character c = null;

		for(int i = 0; i < getI(); i++) {
			c = getCadenaInicial().charAt(i);
			operaciones.add(new OperacionImpl("E", c, i));
		}
	}

	return new SolucionTransCadImpl(operaciones);
}
```

c. *int geNumeroDeAlternativas();*

```
public int getNumeroDeAlternativas () {
	int alternativas = 1;

	if(!getCaracterI().equals(getCaracterJ())) {
		alternativas = 3;
	}

	return alternativas;
}
```

d. *int getNumeroSubProblemasAnd(String a);*

```
public int getNumeroSubProblemasAnd(String a) {
	return 1;
}
```

e. *ProblemaPD<SolucionTransCad, String> getSubProblemaAnd(String a, int i);*

```
public ProblemaPD<SolucionTransCad, String> getSubProblemaAnd(String a, int i) {
	ProblemaPD<SolucionTransCad, String> p = null;

	if(a.equals("N") || a.equals("C")) {
		p = new ProblemaTransCadPDImpl(getI() - 1, getJ() - 1);
	}
	else if(a.equals("E")) {
		p = new ProblemaTransCadPDImpl(getI() - 1, getJ());
	}
	else if(a.equals("A")) {
		p = new ProblemaTransCadPDImpl(getI(), getJ() - 1);
	}
	else {
		throw new SituacionIlegal("La alternativa no es válida.");
	}

	return p;
}
```

f. *SolucionTransCad combinaSolucionesAnd(String a, SolucionTransCad… ls);*

```
public SolucionTransCad combinaSolucionesAnd(String a, SolucionTransCad... ls) {
        SolucionTransCad s = ls[0].clone();

        if(a.equals("C")) {
                s.getOperaciones().add(new OperacionImpl(a, getCaracterJ(), getI() - 1));
        }
        else if(a.equals("E")) {
                s.getOperaciones().add(new OperacionImpl(a, getCaracterI(), getI() - 1));
        }
        else if(a.equals("A")) {
                s.getOperaciones().add(new OperacionImpl(a, getCaracterJ(), getI() - 1));
        }
        else if(!a.equals("N")) {
                throw new SituacionIlegal("La alternativa no es válida.");
        }

        return s;
}
```