Apuntos de Clase de Auxiliatura

3) TDA's y sus funciones.

Tipo: PUNTO[G] / G es un real

Funciones:

crear:	**GxG**	**→**	**PUNTO[G]**
obtenerX:	**PUNTO[G]**	**→**	**G**
obtenerY:	**PUNTO[G]**	**→**	**G**
establecerX:	**PUNTO[G]xG**	**→**	**PUNTO[G]**
establecerY:	**PUNTO[G]xG**	**→**	**PUNTO[G]**
mover:	**PUNTO[G]xPUNTO[G]**	**→**	**PUNTO[G]**
distancia:	**PUNTO[G]xPUNTO[G]**	**→**	**G**

Tipo: POLIGONO[PUNTO[G]] / I pertenece a los Naturales positivos.

Im

Funciones:

crear:	**LISTA[PUNTO[G]]**	**→**	**POLIGONO[PUNTO[G]]**
area:	**POLIGONO[PUNTO[G]]**	**→**	**double (real)**
perimetro:	**POLIGONO[PUNTO[G]]**	**→**	**double (real)**
segmentoMasLargo	**POLIGONO[PUNTO[G]]**	**→**	**SEGMENTO[PUNTO[G]]**
cambiarPunto	**POLIGONO[PUNTO[G]]xIxPUNTO[G]**	**→**	**POLIGONO[PUNTO[G]]**
obtenerPunto	**POLIGONO[PUNTO[G]]xI**	**→**	**PUNTO[G]**
numeroPuntos	**POLIGONO[PUNTO[G]]**	**→**	**int (entero)**
existeInterseccion	**POLIGONO[PUNTO[G]]**	**→**	**boolean**
existePuntosRepetidos	**POLIGONO[PUNTO[G]]**	**→**	**boolean**

Tipo: SEGMENTO[PUNTO[G]]

crear:	**PUNTO[G]xPUNTO[G]**	**→**	**SEGMENTO[PUNTO[G]]**
intersecta:	**SEGMENTO[PUNTO[G]]xSEGMENTO[PUNTO[G]]**	**→**	**boolean**

4) Implementacion TDA (solo variables de instancia y metodos) , sin implementacion de metodos.

```
//Implementacion sin uso de genericas
public class Punto{
	int x;
	int y;
	public Punto(T xo,int yo){
		x=xo;  y=yo;
	}
	public int obtenerX(){return x;}
	public int obtenerY(){return y;}
	public void establecerX(T _x){ x=_x; }
	public void establecerY(T _y){ x=_y; }
	public void mover(Punto<T> otro){ ...}
	public int distancia(){...}
}
public class Poligono{
	private Punto[] puntos;
	public Poligono(Punto[] _puntos){...}
	public double area(){...}
	public double perimetro(){...}
	public double perimetro(){...}
	public Segmento	segmentoMasLargo(){...}
	public Segmento	cambiarPunto(int posicion, Punto<T> punto){...}
	public Punto  obtenerPunto(int posicion){...}
	public int numeroPuntos(){...}
	public int existeInterseccion(){...}
	public int existePuntosRepetidos(){...}
}
public class Segmento{
	private Punto origen;
	private Punto destino;
	public Segmento(Punto o, Punto d){...}
	public boolean intersecta(Segmento otro){...}
}

//Aqui otra version de implementacion usando genericas en java
public class Punto <T extends Number>{
	T x;
	T y;
	public Punto(T xo,T yo){
		x=xo;  y=yo;
	}
	public T obtenerX(){return x;}
	public T obtenerY(){return y;}
	public void establecerX(T _x){ x=_x; }
	public void establecerY(T _y){ x=_y; }
	public void mover(Punto<T> otro){ ...}
	public T distancia(){...}
}
public class Poligono<T extends Number> {
	private Punto<T>[] puntos;
```

```
        public Poligono(Punto<T>[] _puntos){...}
        public double area(){...}
        public double perimetro(){...}
        public double perimetro(){...}
        public Segmento<T>    segmentoMasLargo(){...}
        public Segmento<T>    cambiarPunto(int posicion, Punto<T> punto){...}
        public Punto<T>       obtenerPunto(int posicion){...}
        public int numeroPuntos(){...}
        public int existeInterseccion(){...}
        public int existePuntosRepetidos(){...}
}
public class Segmento<T extends Number>{
        private Punto<T> origen;
        private Punto<T> destino;
        public Segmento(Punto<T> o, Punto<T> d){...}
        public boolean intersecta(Segmento<T> otro){...}
}
```

5) Implementacion de algunos contratos para la clase Poligono, (en la evaluacion se deve hacer lo mismo para todas las clases y todas las funciones).

```
public class Poligono{
        private Punto[] puntos;
        public Poligono(Punto[] _puntos){
                assert (_puntos.length >2) :"error";
                puntos=_puntos;
                assert (invariante()):"error";

        }
        ...
        public Segmento       cambiarPunto(int posicion, Punto<T> punto){
                assert (posicion>=0 && posicion< numeroPuntos()):" error ";
                ...
                assert invariante():" error ";
        }
        public Punto<T>       obtenerPunto(int posicion){
                assert (posicion>=0 && posicion< numeroPuntos()):" error ";
                ...
                assert invariante():" error ";
        }
        ...
        public boolean invariante(){
                return (obtenerArea()>0) &&
                                (numeroPuntos()>2) &&
                                (!existeInterseccion()) &&
                                (!existePuntoRepetido());
        }
}
```