TERCER PARCIAL – 10 de Noviembre de 2023

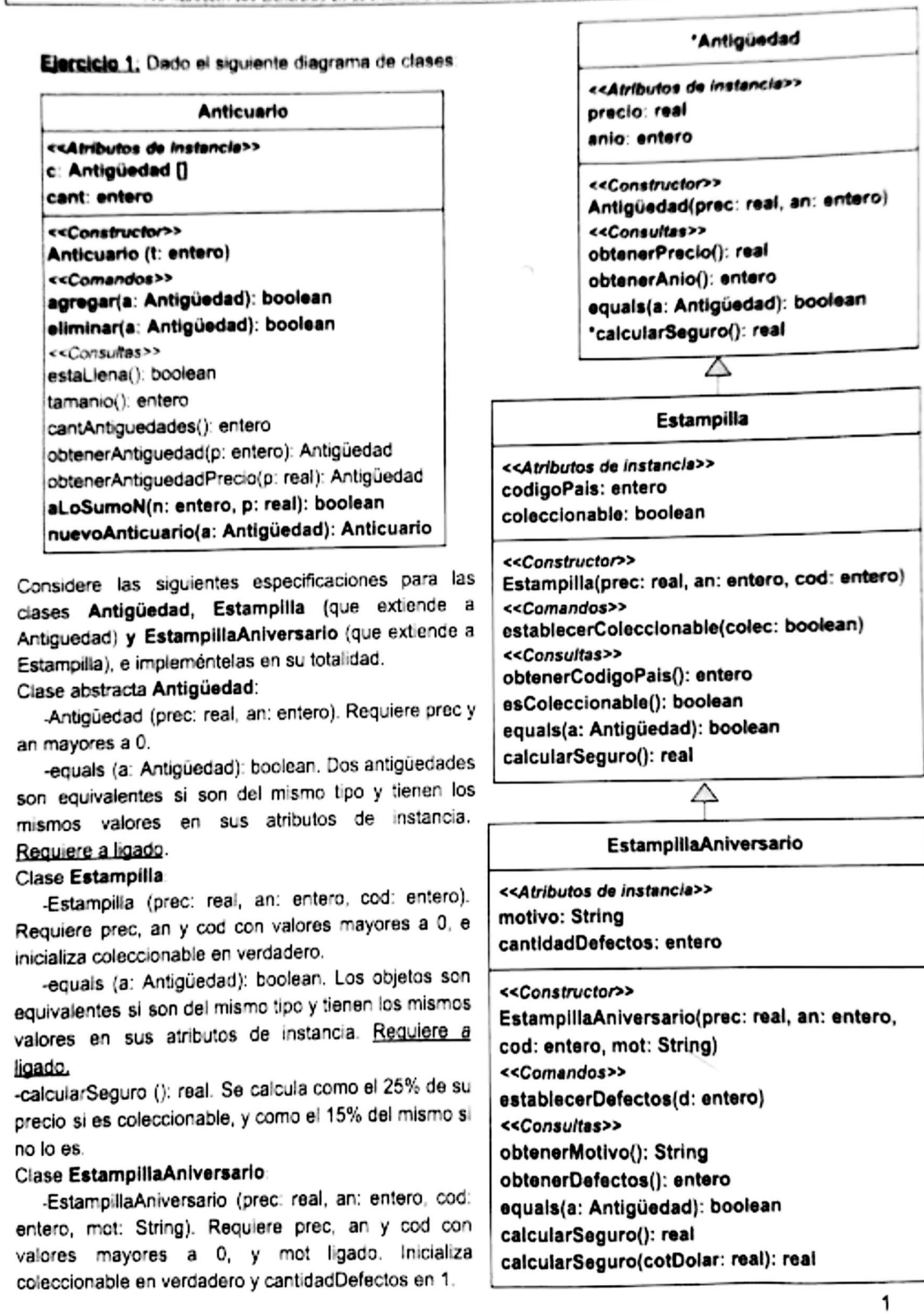
APELLIDO Y NOMBRE: LU: CANT. DE HOJAS [1/ENUNCIADO]:

Importante: La interpretación del enunciado es parte del examen.

LA EVALUACIÓN SE REALIZARÁ CONSIDERANDO LA CORRECTITUD, LEGIBILIDAD Y EFICIENCIA DE LAS SOLUCIONES PROPUESTAS.

NO RESUELVA LOS EJERCICIOS EN EL ENUNCIADO. REALICE LOS PROBLEMAS EN HOJAS SEPARADAS.

Ejercicio 1. Dado el siguiente diagrama de clases



Considere las siguientes especificaciones para las clases **Antigüedad, Estampilla** (que extiende a Antiguedad) **y EstampillaAniversario** (que extiende a Estampilla), e impleméntelas en su totalidad.

Clase abstracta **Antigüedad**:

-Antigüedad (prec: real, an: entero). Requiere prec y an mayores a 0.

-equals (a: Antigüedad): boolean. Dos antigüedades son equivalentes si son del mismo tipo y tienen los mismos valores en sus atributos de instancia. Requiere a ligado.

Clase **Estampilla**:

-Estampilla (prec: real, an: entero, cod: entero). Requiere prec, an y cod con valores mayores a 0, e inicializa coleccionable en verdadero.

-equals (a: Antigüedad): boolean. Los objetos son equivalentes si son del mismo tipo y tienen los mismos valores en sus atributos de instancia. Requiere a ligado.

-calcularSeguro (): real. Se calcula como el 25% de su precio si es coleccionable, y como el 15% del mismo si no lo es.

Clase **EstampillaAniversario**:

-EstampillaAniversario (prec: real, an: entero, cod: entero, mot: String). Requiere prec, an y cod con valores mayores a 0, y mot ligado. Inicializa coleccionable en verdadero y cantidadDefectos en 1.

-establecerDefectos (d: entero). Requiere d >= 0.

-equals (a: Antigüedad): boolean. Los objetos son equivalentes **si son del mismo tipo** y tienen los mismos valores en sus atributos de instancia. Compara el motivo por equivalencia.

-calcularSeguro (): real. Se calcula igual que una estampilla común, con un incremento **del 50% en su** valor si tiene algún defecto.

-calcularSeguro(cotDolar: real): real. Se calcula como en el valor de un seguro de estampilla aniversario dividido la cotización del Dolar (cotDolar). Requiere cotDolar >0.

La clase **Anticuario** representa una colección no ordenada de Antigüedades. Implemente el encabezado de la clase con su constructor, los atributos de instancia, y aquellos métodos que se encuentran en negrita (agregar, eliminar, aLoSumoN y nuevoAnticuario):

-Anticuario (t: entero). Requiere t > 0.

-agregar (a: Antigüedad): boolean. Si la colección no se encuentra llena, agrega la Antigüedad *a* y retorna true, de lo contrario retorna false. Requiere a ligada y no presente en la colección.

-eliminar (a: Antigüedad): boolean. Si la Antigüedad a está presente la elimina de la colección y retorna true, de lo contrario retorna false. Requiere a ligada. Compara por equivalencia.

-obtenerAntiguedad (p: entero): Antigüedad. Retorna la antigüedad en la posición p. Requiere posición p válida y Anticuario no vacío.

-obtenerAntiguedadPrecio (p: real): Antigüedad. Retorna la primera antigüedad con precio igual a *p*, o nulo si no encuentra ninguna. Requiere *p* > 0.

-aLoSumoN (n: entero, p: real): boolean. Retorna verdadero si y sólo si existen a lo sumo n antigüedades con precio de seguro mayor o igual a *p*. Requiere *p* >= 0, que la colección no esté vacía, y 0 < n < cantAntiguedades().

-nuevoAnticuario (a: Antigüedad): Anticuario. Retorna un nuevo Anticuario con el mismo tamaño que el que recibe el mensaje, y que conserve sólo aquellos elementos cuyo precio sea mayor al de a. Requiere a ligado.

Ejercicio 2.

```
public abstract class Uno {
  protected String a;

  public Uno(){     a = "uno";  }

  public String m(){ return "m en Uno";}

public String n(String x){
return "n en Uno" + "x:" + x; }

public String p(){
return "p en Uno" + m();  }

public abstract String q();

}
```

```
public class Dos extends Uno {
  protected String b;

  public Dos(){  b = "dos";  }

  public String m(int x){
  return "m en Dos" + "x:" + x; }

public String n(int x){
return "n en Dos" + "x:" + x; }

public String q(){
  return "q en Dos" + "a:" + a + "b:" + b;  }
}
```

```
class Tres extends Uno {
  protected int c;

public Tres(){  c= 3;  }

public String m(){return "m en Tres" + "a:" + a; }

 public String n(){ return "n en Tres" + "a:" + a; }

  public String p(){
    String s = super.p();
    return s + "p en Tres";  }

  public String q(){
    return "q en Tres" + "c" + c + "a" + a;  }
}
```

Indique y justifique adecuadamente cuáles de las siguientes instrucciones provocarán errores, e indique de qué tipo son (si de compilación o de ejecución). Siempre que sea posible, muestre cuál será la salida por consola.

```
Uno u1, u2, u3, u4;

Dos d1, d2, d3, d4;

Tres t1, t2, t3;

1.  u1 = new Dos();
2.  u2 = new Tres();
3.  u3 = new Uno();
4.  d1 = u1;
5.  d2 = new Tres();
6.  u4 = null;
7.  t1 = new Tres();
8.  d3 = u4;
9.  d4 = new Dos();
10. t2 = (Tres) u2;
11. t3 = new Tres();
12. System.out.println(u2.q());
13. System.out.println(u1.m(4));
14. System.out.println(u4.m());
15. System.out.println(d4.n("dos"));
16. System.out.println(t3.p());
```