Estructura de Datos 20/21

Práctica 2

Programación Orientada a Objetos (POO) para las Estructuras de Datos

Objetivos

- Poner en práctica lo visto y aprendido en el tema 2 de teoría
- Realizar ejercicios relacionados con la POO para las Estructuras de Datos
- Reforzar habilidades de aprender a aprender

Descripción del laboratorio

Realizar y entregar los siguientes ejercicios.

Ejercicio 1 - Aliases

¿Qué imprime el siguiente programa? Razona tu respuesta. Puedes ejecutar el código en IntelliJ IDEA.

```
public class Aliases {
   public static void main(String[ ]args) {
     Persona x = new Persona();
     Persona y = new Persona();
     x.nom = "Joan";
     y.nom = "Alberto";
     Persona z;
     z = x;
     x = y;
     x.nom = "Marc";
     System.out.println("Nom en x es:“ + x.nom);
     System.out.println("Nom en y es:“ + y.nom);
     System.out.println("Nom en z es:“ + z.nom);
   }
 }
```

Ejercicio 2 – Encapsulamiento

Contesta a las siguientes preguntas:

a) ¿Qué bloque puede acceder al atributo `nameDay`? Razona tu respuesta.

```
public class Ej2{
    public String nameDay;
        public static void main(String[] args) {
             //Bloque 1
              nameDay = "Monday";
              System.out.println("Today is: " + nameDay);
              //Bloque 2
              Ej2 ej2 = new Ej2();
              ej2.nameDay = "Monday";
              System.out.println("Today is: " + ej2.nameDay);
        }
 }
```

b) ¿Desde la case C, se puede imprimir el atributo **x** de la clase A? Razona tu respuesta.

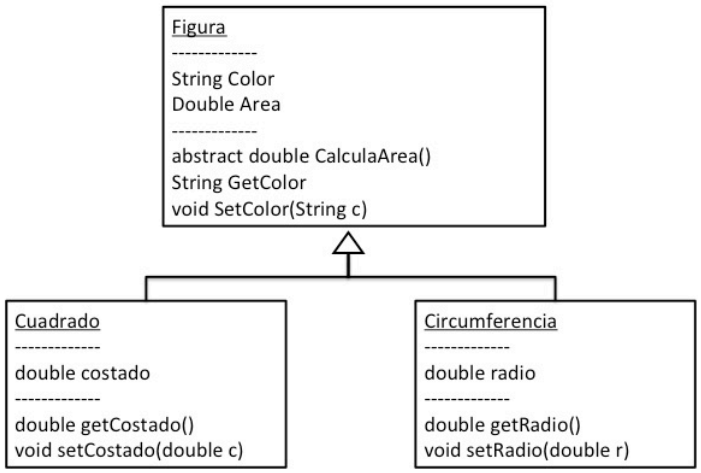
```
package unPaquet;
public class A {
    private int x;
    public A() {
       x=1;
    }
}
package unPaquet;
public class C {
    A a;
    public C(){
        a=new A();
    }
    public void metodeC(){
    System.out.println("El valor de a es:" + a.x);
    }
```

c) Si queremos mantener el atributo de la clase A como privado, ¿qué modificaciones tenemos que hacer en la clase A para poder imprimir el valor del atributo **x** de la clase C? Razona tu respuesta.

```
package unPaquete;
 public class A {
     private int x;
     public A() {
       x=1;
     }
 }
  package unPaquete;
public class C {
    A a;
    public C(){
        a=new A();
    }
    public void metodoC(){
    System.out.println("el valor de a as:" + <código>);
    }
```

Ejercicio 3 – Herencia

Implementa el siguiente diseño, considerando que la clase Figura es abstracta.



Ejercicio 4 – Genéricos

Implementa un método genérico que imprima todos los elementos de una lista con un `Iterator`. Puedes probar el método en el siguiente escenario:

```
import java.util.*;
public class Ejercicio4{
  public static void main (String [] args){
    Professor pf1 = new Professor ("Professor_Sergio", "1234-D", "3.14");
    Professor pf2 = new Professor ("Professor_Xavi", "1235-D", "3.15");
    ArrayList<Professor> list_prof = new ArrayList<Professor>();
    list_prof.add(pf1);
    list_prof.add(pf2);
    Lector l1 = new Lector ("Lector_Sergio", "1234-D", "GRIHO");
    Lector l2 = new Lector ("Lector_Josep", "1234-E", "GTI”);
    LinkedList<Lector> list_lec = new LinkedList<Lector>();
    list_lec.add(l1);
    list_lec.add(l2);
     printList(list_prof);
     printList(list_lec);
}
//añadir vuestro método aqui
```

Ayuda:

- Diseñar la clase Persona, Professor y Lector.
- La clase Persona tiene 2 atributos (nombre, DNI). Implementa set y get para cada atributo
- Un Professor (catedrático, en Inglés) es una Persona, con un despacho. Implementa set y get para los atributos utilizando correctamente la herencia.
- Un Lector es una Persona, con un grupo de investigación. Implementa set y get para los atributos utilizando correctamente la herencia.
- La clase Persona tiene un método `toString()` definido de la siguiente manera:

```
public String toString()
{
    return "Nom: " + nom +" DNI: " + dni;
}
```

- Implementa el método toString() de Professor y Lector utilizando herencia – es decir, a partir del método toString() de Persona.

Ejercicio 5 – Genéricos y Comparable

Implementa un método sort (ordenar) genérico que permita ordenar un `ArrayList` de Personas según su DNI (que será un String) y una `LinkedList` de Coches según su matrícula (que será un String). No se puede utilizar el método sort de Collections.

Las Personas y los Coches serán del tipo `Comparable`, que es una interfaz genérica de la JCF. Consulta la documentación de clase en el siguiente enlace:

https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Comparable.html

Los coches tienen matrícula. Diremos que dos coches son iguales si tienen la misma matrícula.

Las personas tiene nombre y DNI. Compararemos personas según su DNI.

El juego de pruebas para este ejercicio es el siguiente:

```
import java.util.*;
public class Exercici2{
  public static void main (String [] args){
    Persona person1 = new Persona("Sergio", "99999999X");
    Persona person2 = new Persona ("Josep", "88888888P");
    //mas personas
    ArrayList<Persona> arr_per = new ArrayList<Persona>();
    arr_per.add(person1);
    arr_per.add(person2);
    // añadir mas personas
    sort(arr_per);
    //imprimir lista de Personas ordenades (opcional)
    Cotxe coche1 = new Coche ("L-1234");
    Cotxe coche2 = new Coche ("Z-1234");
    //mas coches
    LinkedList<Cotxe> arr_ctx = new LinkedList<Coche>();
    arr_ctx.add(coche1);
    arr_ctx.add(coche2);
    //añadir mas coches
    sort(arr_ctx);
    //imprimir lista de coches ordenados (opcional)
}
//el vostre mètode sort aquí
```

Pistas:

- Decir que Coches y Personas son del tipo Comparable significa que implementan una interfaz, en este caso, la interfaz Comparable
- Para ordenar, puedes utilizar el algoritmo de BubbleSort, o cualquier algoritmo similar de ordenación
- La cabecera del método que os pido implementar es la siguiente:

```
public static <T extends Comparable> void sort (List<T> l)
```

Entrega

Fecha: ver la tarea correspondiente en el Campus Virtual

Material: un documento comprimido (ZIP) con los siguientes ficheros:

- Un documento DOC / RTF / PDF con las respuestas a los Ejercicios 1 y 2. Extensión máxima: 1 página.
- Un ZIP con los ficheros .java del Ejercicio 3
- Un ZIP con los ficheros .java del Ejercicio 4
- Un ZIP con los ficheros .java del Ejercicio 5

Criterios (generales) de evaluación

- Práctica no entregada o entregada fuera de plazo = 0
- Entrega parcial, <= 3
- Entrega completa, entre 4 y 10. Se valorará el diseño y claridad del código, y el razonamiento de las respuestas.