METODOLOGÍA DE LA PROGRAMACIÓN

Guion de prácticas de laboratorio.

Profesores: Jorge Puente, Alfredo Alguero, Eugenia Díaz, Juan Luis Mateo y Ramiro Varela

TEMA 1. Programación Orientada a Objetos. HERENCIA

Material para dejar en el Campus Virtual: 1 fichero zip con 6 proyectos que contienen las siguientes clases con sus correspondientes ejemplos de prueba

- **`00_Bicycle`**: clase **`Bicycle`**
- **`Fig_9_5_CommissionEmployee`**: clase **`CommisionEmployee`**
- **`Fig_9_6_BasePlusCommissionEmployee`**: clase **`BasePlusComissionEmployee`**
- **`Fig_9_8_BasePlusCommissionEmployee`**: Clases **`CommisionEmployee`** y **`BasePlusCommissionEmployee`** definidas mediante herencia
- **`Circles`**
- **CommissionAndBasePlusEmployees**
- **`(segundo bloque...)`**

PRÁCTICA 1.1. Uso de clases simples

Objetivo: Familiarizarse con las clases básicas **`Bicycle`**, **`CommisionEmployee`** y **`BasePlusCommissionEmployee`**, y saber hacer algunas modificaciones en clases que ya están programadas.

Ejercicio 1.1.1. Uso y modificación de la clase **`Bicycle`**

a) Accede al proyecto 00_Bicycle 1.0
b) Modifica la clase **`Bicycle`** para que:
   i. Cumpla los requisitos del principio de encapsulación. Es decir, que cada dato tenga un observador y un único modificador, y que el resto de los métodos (incluidos los constructores) accedan a los datos a través de estos observadores y modificadores.
   ii. Que los atributos se inicien exclusivamente en los constructores. Los atributos de instancia no deben iniciarse en la declaración (excepto constantes **`final`**).
c) Añade operaciones al ejemplo de uso en el método **`main`**(). Por ejemplo, crea otra bicicleta con la misma **`gear`** que **`bike1`** y con el doble de **`speed`** de **`bike2`**.
d) Añade un constructor de copia y pon algún ejemplo de uso.
e) Implementar el método **`toString()`** en la clase **`Bicycle`** y utilízalo para mostrar objetos en modo texto.
f) Verifica el correcto funcionamiento del código pasando las pruebas unitarias de la carpeta **test.**

> **Recuerda:** Para lanzar las pruebas, con el ratón sobre el icono de la carpeta **test**, en el menú contextual, **Run As → JUnit Test.**

Ejercicio 1.1.2. Añadir operaciones a los ejemplos de uso de **`CommissionEmployee`**

a) Accede al proyecto Fig_9_5_CommissionEmployee y ejecuta el ejemplo de prueba que incluye.
b) En el método **main()**, crea un vector de trabajadores de la clase **`CommissionEmployee`**, y muestra los datos del trabajador que más gana.
   i. Para ello debes definir el método:

```
public static CommissionEmployee
     maxEarningsEmployee(CommissionEmployee[] v);
```

c) Un poco más complicado que el anterior. Muestra por consola todos los trabajadores, pero ordenados por su salario de menor a mayor.
   i. Para ello debes definir los siguientes métodos en la clase **CommissionEmployeeMain**:

```
public static void sortEmployees(CommissionEmployee[] v);
public static void showEmployees(CommissionEmployee[] v);
```

d) Verifica el correcto funcionamiento del código pasando las pruebas unitarias de la carpeta **test.**

Ejercicio 1.1.3. Lo mismo que el ejercicio anterior, pero con la clase **`BasePlusCommissionEmployee`**, es decir

a) Accede al proyecto Fig_9_6_BasePLusCommissionEmployee y ejecuta el ejemplo de prueba que incluye.
b) Crea un vector de trabajadores de la clase **`BasePlusCommissionEmployee`**, y muestra los datos del trabajador que más gana.
    i. Para ello debes definir el método:

```
public static BasePlusCommissionEmployee
     maxEarningsEmployee(BasePlusCommissionEmployee[] v);
```

c) Muestra por consola todos los trabajadores, pero ordenados por su salario de menor a mayor.
    i. Para ello debes definir los siguientes métodos en la clase **BasePlusCommissionEmployeeMain**:

```
public static void sortEmployees(BasePlusCommissionEmployee [] v);
public static void showEmployees(BasePlusCommissionEmployee [] v);
```

d) Verifica el correcto funcionamiento del código pasando las pruebas unitarias de la carpeta **test.**

PRÁCTICA 1.2. Creación de proyectos que utilizan más de una clase

Objetivo: Usar proyectos con varias clases, y programar algunas clases sencillas.

Ejercicio 1.2.1. Abrir el proyecto **`CommissionEmployeeAndBasePlusEmployees`**

a) Copiar en el proyecto las clases **`CommissionEmployee`** y **`BasePlusCommissionEmployee`**.

b) En el main() crea dos vectores de trabajadores, uno de cada clase, y calcula y devuelve el sueldo del trabajador que más gana.

Para ello, en **`CommissionAndBasePlusEmployeesMain`**, debes definir el método:

```
public static double
     maxEarnings(CommissionEmployee[] v1,
         BasePlusCommissionEmployee[] v2);
```

c) Un poco más complicado que el anterior. Muestra por consola todos los trabajadores de los dos vectores, pero ordenados por su salario de menor a mayor. Así, los trabajadores de las dos clases pueden aparecer intercalados. Este ejercicio requiere ordenar los dos vectores y luego recorrerlos simultáneamente para ir actualizando el vector v3 con las nóminas ordenadas. Es un algoritmo de "mezcla" de dos estructuras ordenadas.

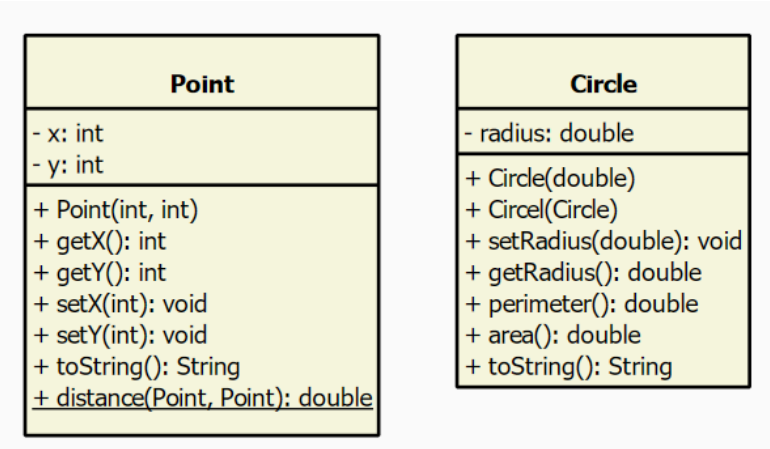
Para ello debes definir el método:

```
public static void
    generateSortedEarnings(CommissionEmployee[] v1,
        BasePlusCommissionEmployee[] v2, double[] v3);
```

Nota: El array v3 debe tener como tamaño la suma de longitudes de v1 y v2.

Ejercicio 1.2.2. Programar dos clases simples **`Circle`** y **`Point`** (iguales o similares a las que se utilizaron en Introducción a la Programación)

a) En el proyecto **`Circles`** crea dos clases, **`Circle`** y **`Point`,** y un programa que utilice las dos clases, con algunos ejemplos de uso.



Importante: Recuerda que un diagrama de clases es un contrato. Esto significa que debes implementar exactamente los nombres, visibilidades, cantidad y orden de argumentos de las clases, atributos y métodos como se indica en el diagrama. No alteres estos elementos, ya que son esenciales para garantizar coherencia con el diseño.

b) La clase **`Circle`** tiene como único dato el **`radius`** y debe tener métodos para ver y modificar el radio, calcular la superficie, así como los constructores apropiados: al menos un constructor a partir de un número real y el constructor de copia.

c) La clase **`Point`** tiene dos datos **`x`** e **`y`** enteros que representan las coordenadas en el plano. Debe incluir un constructor a partir de dos enteros y los métodos observadores y modificadores apropiados, así como un método para calcular la distancia de un punto a otro punto.

d) Verifica el correcto funcionamiento del código pasando las pruebas unitarias de la carpeta **test**.

PRÁCTICA 1.3. Relaciones de composición

Objetivo: Entender bien la relación de composición para distinguirla después de la relación de herencia.

Ejercicio 1.3.1. Define la clase **`BasePlusCommissionEmployee`** por composición de **`CommissionEmployee`**

a) Crea un proyecto nuevo a partir de una copia del proyecto **`Fig_9_6_BasePlusCommissionEmployee`**

b) Copia en el nuevo proyecto la clase **`CommissionEmployee`** de **`Fig_9_5_CommissionEmployee`**

c) Redefine la clase **`BasePlusCommissionEmployee`** con solo dos atributos:
   - un atributo de tipo **`CommissionEmployee`**
   - un atributo con su salario base.

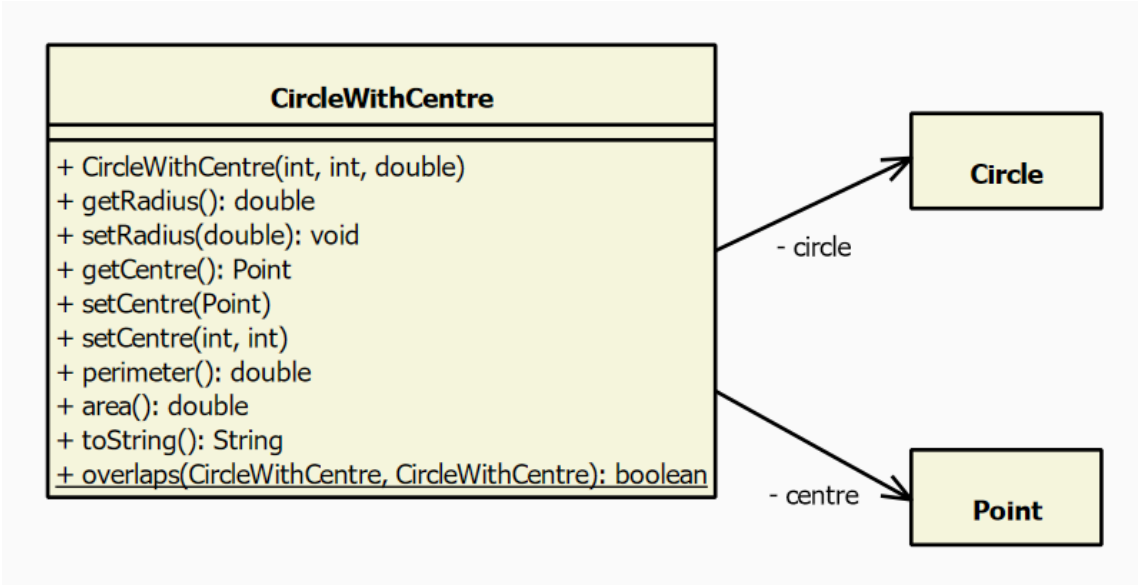
   Define adecuadamente los métodos para que la clase mantenga la misma funcionalidad que en las implementaciones anteriores.

d) Comprueba que los ejemplos de uso de la práctica anterior funcionan y producen los mismos resultados. Comprueba que todas las pruebas unitarias de la carpeta **test** se siguen ejecutando correctamente.

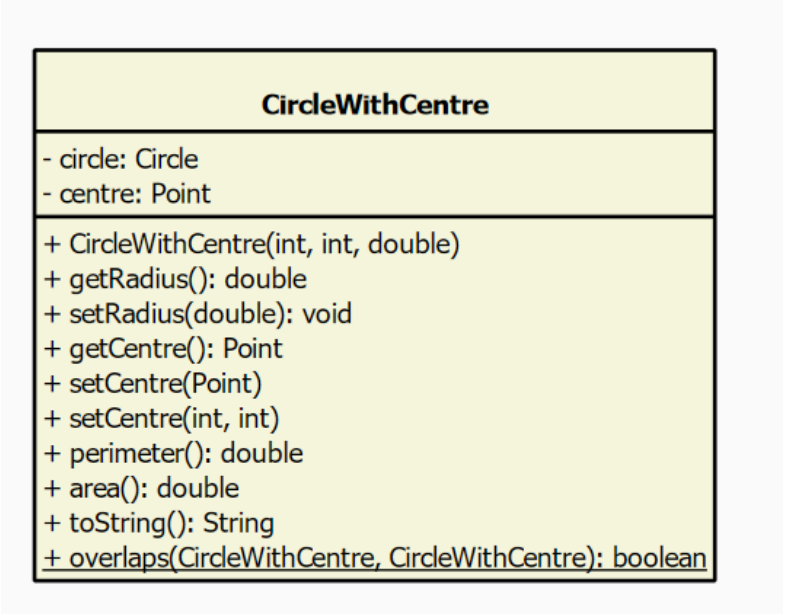
e) Comenta las ventajas y/o desventajas que tienen las dos formas de definir las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** , es decir definirlas de forma independiente, o bien utilizando la relación de composición.

Ejercicio 1.3.2. Define la clase **`CircleWithCentre`** a partir de las clases **`Circle`** y **`Point`** mediante relación de composición.

f) En el proyecto **`Circles`**, define la clase **`CircleWithCentre`** haciendo la interpretación de que un círculo con centro “tiene un” círculo y “tiene un” centro. Incluye los métodos observadores y modificadores, y los constructores apropiados.
   Observa los dos gráficos siguientes que representan la estructura de la clase. Se trata de representaciones equivalentes, pero la segunda muestra de forma más clara la relación de composición.



Diagramas Equivalentes



a) Incluye, en las clases apropiadas, métodos para modificar la posición del centro, para aumentar o disminuir el radio, para calcular el perímetro y el área de un círculo y para indicar si un círculo se solapa con otro o no.

b) Verifica el correcto funcionamiento del código pasando las pruebas unitarias de la carpeta **test**.

PRÁCTICA 1.4. Relaciones de HERENCIA

Objetivo: Entender bien los conceptos relacionados con la herencia, y distinguir la herencia de la composición.

Ejercicio 1.4.1. Uso y modificación de las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** definidas mediante herencia.

Cada vez que modifiques un método, coloca la implementación anterior en un comentario para poder compararla luego con la nueva implementación.

a) Accede al proyecto Fig_9_8_BasePlusCommissionEmployee y comprueba que los programas de prueba de las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** de las prácticas anteriores funcionan igual cuando estas clases están definidas a través de la relación de herencia.
b) Modifica el constructor de la subclase **`BasePlusCommissionEmployee`** de modo que no utilice la llamada **`super(first,last,ssn,sales,rate)`** para invocar al constructor de la superclase. Para ello tiene que asignar valores a los 5 datos definidos en la superclase de la forma apropiada.
c) En la implementación del método **`earnings`** de la subclase, cambia las llamadas a los métodos observadores **`getCommissionRate()`** y **`getGrossSales()`** por los nombres de los campos de datos correspondientes. Observa el problema que se produce. Cambia el calificador **`private`** por **`protected`** en los campos de la superclase y observa que todo funciona otra vez.
d) Modifica la implementación del método **`earnings`** de la subclase de modo que se utilice **`super`** para invocar al método de la superclase.
e) Toma nota de las tres formas de implementar el método **`earnings`** en la subclase (la original, la del apartado c) y la del apartado d)) y escribe un pequeño comentario sobre las ventajas y/o inconvenientes de cada uno de ellos, y cuál te parece la forma más razonable de implementarlo.

Ejercicio 1.4.2. Define la clase **`CircleWithCentre`** a partir de las clases **`Circle`** y **`Point`** utilizando relaciones de herencia.

a) Duplica el proyecto **`Circles`** con nombre **`CirclesByCompositionAndHierarchy.`**
b) Redefine la clase **`CircleWithCentre`** considerando que un círculo con centro “es un” círculo que “tiene un” centro. La clase debe incluir la misma funcionalidad que la definida en la práctica anterior (Ejercicio 1.3.2). Comprueba que las pruebas unitarias de la práctica anterior funcionan correctamente con esta nueva versión basada en herencia y composición.

DISTRIBUCIÓN TEMPORAL

Sesión 1: PRACTICA 1.1.

Sesión 2. PRACTICA 1.2.

Sesión 3. PRÁCTICA 1.3.

Sesión 4. PRÁCTICA 1.4.

Sesión 5. PRÁCTICA 1.5. (segundo bloque)

Sesión 6. PRÁCTICA 1.6. (segundo bloque)

Sesión 7. PRÁCTICA 1.7. (segundo bloque)

Sesión 8: PRÁCTICA 1.8. (segundo bloque)

Sesión 9. PRÁCTICA 1.9. (segundo bloque)