Práctica Calificada 4

Curso: CC201 **Ciclo: 2018.2**

Para la preguntas 1 y 2: implemente la clase `Pc4` tal que contenga el método `main` de donde se llamen –iterativamente hasta seleccionar la opción de salida– los métodos de las otras clases. Para ello, complete en los puntos suspensivos:

```
import java.util.Scanner;
public class Pc4 {
   public static void main(String[] args){
      Scanner entrada = new Scanner(System.in);
      int opcion;
      do{
         System.out.printf("%n%s%n%s%n%s%n%s%n%s%n%n%s",
         "Menu", "0.- salir", "1.- pregunta 1",
         "2.- pregunta 2"
         "Seleccione una de las opciones: ");
         opcion = entrada.nextInt();
         // Complete aqui.....................
      } while(opcion != 1);
   }
   public static void pregunta1(){ // pregunta 1
   }
   public static void pregunta2(){ // pregunta 2
   }
}
```

1. (6 ptos) En clase se estudió una jerarquía de herencia en donde la clase `EmpleadoBaseMasComision` heredó de la clase `EmpleadoPorComision`. Sin embargo, no todos los tipos de empleados son `EmpleadoPorComision`. En este ejercicio, creará una superclase `Empleado` más general que extraiga los atributos y comportamientos de la clase `EmpleadoPorComision` que son comunes para todos los objetos `Empleado`. Los atributos y comportamientos comunes de todos los objetos `Empleado` son: `primerNombre`, `apellidoPaterno`, `dni`, `obtenerPrimerNombre`, `obtenerApellidoPaterno`, `obtenerDni` y una parte del método `toString`. Cree una nueva superclase `Empleado` que contenga estas variables y métodos de instancia, además de un constructor. A continuación, vuelva a escribir la clase `EmpleadoPorComision` como una subclase de `Empleado`. La clase `EmpleadoPorComision` debe contener sólo las variables y métodos de instancia que no se declaren en la superclase `Empleado`, estos son: `ventasBrutas`, `tarifaComision`

y los métodos establecer y obtener para manipular estas dos variables. El constructor de la clase `EmpleadoPorComision` debe invocar al constructor de la clase `Empleado` y el método `toString` de `EmpleadoPorComision` debe invocar al método `toString` de `Empleado`. El método `establecerVentasBrutas` debe asegurarse de que `ventasBrutas` sea mayor o igual a 0, `establecerTarifaComision` debe asegurar que el valor de `tarifaComision` esté entre 0 y 1, y el constructor debe verificar el correcto estado de `ventasBrutas` y `tarifaComision`, así como lo hacen los métodos `establecerVentasBrutas` y `establecerTarifaComision`. Una vez que complete estas modificaciones, pruebe dichas modificaciones creando e 'imprimiendo' un objeto de la clase `EmpleadoPorComision`.[1]

2. (6 ptos) Otros tipos de objetos `Empleado` podrían incluir objetos `EmpleadoAsalariado` que reciban un salario semanal fijo, `TrabajadoresPiezas` a quienes se les pague por el número de piezas que produzcan, o `EmpleadosPorHoras` que reciban un sueldo por horas con tiempo y medio (1.5 veces el sueldo por horas) por las horas trabajadas que sobrepasen las 40 horas. Cree la clase `EmpleadoPorHoras` que herede de la clase `Empleado` (ejercicio de arriba) y tenga la variable de instancia `horas` (de tipo `double`) que represente las horas trabajadas, la variable de instancia `sueldo` (de tipo `double`) que represente los sueldos por hora, un constructor que reciba como argumentos el primer nombre, el apellido paterno, el número de dni, el sueldo por horas y el número de horas trabajadas, métodos establecer y obtener para manipular las variables `hora` y `sueldo`, un método `ingresos` para calcular los ingresos de un `EmpleadoPorHoras` con base en las horas trabajadas, y un método `toString` que devuelva la representación `String` del `EmpleadoPorHoras`. El método `establecerSueldo` debe asegurarse de que `sueldo` sea mayor o igual a 0, y `establecerHoras` debe asegurar que el valor de `horas` esté entre 0 y 168 (el número total de horas en una semana). Finalmente, cree e 'imprima' un objeto de la clase `EmpleadoPorHoras`.[2]

3. (4 ptos) Dibuje un diagrama de UML de una jerarquía de herencia para los estudiantes en una universidad, de manera similar a la jerarquía que se muestra en la figura 9.2. Use a `Estudiante` como la superclase de la jerarquía, y después extienda `Estudiante` con las clases `EstudianteNoGraduado` y `EstudianteGraduado`. Siga extendiendo la jerarquía con el mayor número de niveles que sea posible. Por ejemplo, `EstudiantePrimerAnio`, `EstudianteSegundoAnio`, `EstudianteTercerAnio` y `EstudianteCuartoAnio` podrían extender a `EstudianteNoGraduado`, y `EstudianteDoctorado` y `EstudianteMaestria` podrían ser subclases de `EstudianteGraduado`.[3]

[1] Ejercicio 9.14 de [1]
[2] Ejercicio 9.15 de [1]
[3] Ejercicio 9.5 de [1]

4. (4 ptos) Dibuje un diagra de UML de una jerarquía de herencia para las clases `Cuadrilatero`, `Trapezoide`, `Paralelogramo`, `Rectangulo` y `Cuadrado`. Use `Cuadrilatero` como la superclase de la jerarquía. Cree y use una clase `Punto` para representar los puntos en cada figura. Agregue todos los niveles que sea posible a la jerarquía. Especifique las variables de instancia y los métodos para cada clase. Las variables de instancia `private` de `Cuadrilatero` deben ser los pares de coordenadas x-y y para los cuatro vértices del `Cuadrilatero`, y cada clase, excepto `Cuadrilatero`, debe tener el método `area` que retorne el área de cada objeto (debe emplear sobrescritura).[4]

Referencias

[1] Deitel, P., and Deitel, H. *Java How to Program: Early Objects.* Pearson Education, 2015.

7 de noviembre de 2018

[4] Ejercicio 9.8 de [1]