TRABAJO PRÁCTICO 8

Tema: Tipos primitivos y tipos referencia.

Objetivos:

Que el estudiante logre…

- Diferenciar entre un tipo primitivo, como int o float, y un tipo referencia mediante el cual es posible acceder a un objeto de una clase.
- Crear tipos referencia u objetos de clases para resolver problemas mediante el Paradigma de Programación Orientada a Objetos y la aplicación del lenguaje Java.

Condiciones de desarrollo y autoevaluación del alumno.

- Este trabajo práctico debe realizarse en forma individual.
- La codificación debe llevarse a cabo de manera repetitiva y analítica tantas veces como sea necesario, revisando el significado de cada sentencia hasta que el alumno adquiera seguridad conceptual y práctica.
- La resolución completa de este trabajo práctico, incluyendo diagramas de flujo, deberá pasar a integrar la carpeta de práctica y autoevaluación del alumno.

Recursos Bibliográficos

- Programación en lenguaje Java. Ángel Esteban. Grupo Eidos.
- Java™ Platform Standard Ed. 7. https://docs.oracle.com/javase/8/docs/index.html
- Materiales disponibles en el aula virtual, tales como videos, lecciones, etc.

Materiales

- Video denominado “POO, tipos primitivos, referencias y diagramación UML de clases”.
- Bibliografía disponible en el CUV.
- IDE NetBeans o similar.
- Controles VPL para realizar las entregas.

Tareas a desarrollar para cada enunciado

- ❖ Desarrolle la diagramación de flujo y la codificación en lenguaje Java de los enunciados.
- ❖ Para cada problema implemente una solución en lenguaje de programación Java aplicando POO.
- ❖ Identifique en cada caso las clases que se requieren, atributos y métodos aplicando el DOO.
- ❖ Construya un solo proyecto en el entorno NetBeans en el cual incluya la/s clase/s necesarias para resolver cada problema.
- ❖ La resolución de todos los enunciados deberá estar desarrollada en la carpeta de práctica de la asignatura y los programas en Java correspondientes a cada uno de los enunciados deberán ser completamente desarrollados por el alumno.

Enunciado 1

Estructura de la clase Perro

```
public class Perro {
   // variables miembro
   // constructor/es
   // getters y setters

   public int getEdad(){
      ....   }

   public void ladrar(int x){
      ....   }

   public void ladrar(char x){
      ....   }

   public void ladrar(boolean x){
      ....   }

   public String toString(){
      ....   }

   public void mostrar(){
      // convocar a toString()
      ....   }
}
```

Crear una clase Perro que posea los siguientes atributos: nombre, raza, peso, color, año de nacimiento.

- Crear los métodos getters y setters de la clase Perro.
- Agregar un método get que devuelva la edad del objeto Perro.
- Crear un método void ladrar(…) sobrecargado. Este método debería sobrecargarse en base a varios tipos de datos primitivos, e imprimir distintos tipos de ladridos, aullidos, etc. dependiendo de la versión sobrecargada que se invoque.

Por ejemplo, si se convoca a ladrar con un argumento x de tipo char, entonces se emite un ladrido formado por ese carácter combinado con alguna vocal: “uuu” + x + “uuu” + x

- Incluir la redefinición del método toString para la clase Perro.

El método toString puede usarse en cualquier clase indicando como retorno la concatenación de los valores de las variables miembro, siempre respetando lo requerido en el problema. El método sólo devuelve una cadena String con la concatenación de los valores de las variables miembro. Luego el método toString puede ser convocado por un método mostrar.

- Desarrollar una clase Principal en la cual se haga la creación de dos o tres perros y se utilicen su métodos sobrecargados, se muestren sus valores, y se indique la edad de cada uno.

Enunciado 2

Estructura de la clase Cuenta

```
public class Cuenta {
   // variables miembro
   // constructor/es
   // getters y setters

   public void ingresar(double cantidad){
      ....   }

   public void retirar(double cantidad){
      ....   }

   public void mostrar(){
      // convocar a toString()
      ....   }

   public static void main(String[] args) {
      ....   }
}
```

- Crear una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales).

  El dato titular será obligatorio y cantidad es opcional.
- Crear dos constructores:
  - Un constructor con un solo parámetro, el obligatorio: titular.
  - Un constructor con dos parámetros para titular y cantidad.

  Crear métodos get, set y toString.

  Crear los siguientes métodos:
  - public void ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se modifica cantidad.
  - public void retirar(double cantidad): se retira una cantidad de la cuenta, si restando la cantidad a extraer de la cuenta el resultado es negativo entonces la cantidad de la cuenta pasa a ser 0.
- Desarrollar el método main dentro de la clase Cuenta, crear objetos del tipo Cuenta y probar el funcionamiento.

Enunciado 3

Desarrolla una clase Cafetera con los siguientes atributos:

- capacidadMaxima (la cantidad máxima de café que puede contener la cafetera) y
- cantidadActual (la cantidad actual de café que hay en la cafetera).

Implementa, al menos, los siguientes métodos:

- Constructor predeterminado: establece la capacidad máxima en 1000 (c.c.) y la cantidad actual en cero (cafetera vacía).
- Constructor con la capacidad máxima de la cafetera; inicializa la cantidad actual de café igual a la capacidad máxima.
- Constructor con la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
- Accesores y seteadores.
- llenarCafetera(): hace que la cantidad actual sea igual a la capacidad máxima.
- servirTaza(int): simula la acción de servir una taza con una capacidad indicada que se recibe como parámetro de entrada. Si la cantidad actual de café “no alcanza” para llenar la capacidad de la taza, entonces se sirve lo que quede en la cafetera.
- vaciarCafetera(): pone la cantidad de café actual en cero.
- agregarCafe(int): añade a la cafetera la cantidad de café indicada. Este método requiere verificar cantidades.

Desarrollar un programa en el cual se crea una cafetera con la cantidad de café indicada por el usuario, para luego servir tantas tazas de 240 cc como sea posible. Al finalizar indicar la cantidad de tazas servidas.

Enunciado 4

Desarrollar una clase Televisor con los siguientes atributos: marca, cantidad de canales, estado (prendido/apagado), pulgadas, y canal activo.

Implementar, al menos, los siguientes métodos en la clase Televisor:

- Constructor predeterminado: establece los siguientes valores para cada uno de los miembros…

  marca ---> Samsung — pulgadas ---> 42

  cantidad de canales ---> 120 — canal activo ---> 1

  estado ---> false

- Constructor con parámetros de entrada que indican los valores de marca y pulgadas, el resto de miembros se inicializa con los siguientes valores default:

  cantidad de canales ---> 120 — canal activo ---> 1

  estado ---> false

- Constructor que recibe como entrada la cantidad de canales y permite que el usuario ingrese los valores del resto de las variables miembro, excepto el estado que se indica como apagado (falso).
- Accesores y seteadores.
- cambiarEstado(): modifica el estado del televisor.
- mostrar(): muestra por pantalla los valores de las variables miembro del televisor.
- cambiarCanal(int): recibe el número de canal al cual cambiar. Si ese valor supera a la cantidad de canales del televisor, será necesario restarle tantas veces como sea necesario la cantidad real de canales del televisor hasta que el valor sea menor o igual a la cantidad de canales que posee.
- cambiarCanal(boolean): si el parámetro es true se incrementa el canal, de lo contrario se decrementa el canal. Los valores ciclan en el rango de la cantidad de canales, es decir, si son 200 canales es posible pasar desde el canal 200 al canal 1.

Desarrollar un programa en el cual se crea un Televisor con la cantidad de canales indicada por el usuario.

Posteriormente se enciende el Televisor.

El usuario debe poder pasar de canal en canal en forma ascendente o descendente hasta encontrar el canal deseado por el usuario. Después de ver el canal, el usuario decide si se queda y termina o si vuelve a cambiar.

Finalmente, apagar el Televisor informando cuantas veces se cambió de canal y mostrando la información completa del Televisor.

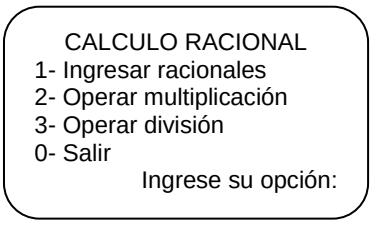
NOTA: en la clase Principal, acompañar cada acción con un mensaje describiendo la acción realizada. Por ejemplo, "Ahora el televisor está sintonizando el canal " e indicar el canal sintonizado mostrando la información del televisor.

Enunciado 5

Crear la clase Racional destinada a realizar aritmética de fracciones. Para su definición tener en cuenta las siguientes indicaciones:

- Utilizar variables enteras para representar las variables miembro de la clase: numerador y denominador. Observar, ¿es posible representar esas variables miembro mediante un tipo de otra clase?
- Proporcionar un método constructor que permita inicializar un objeto de la clase Racional cuando se declara. El constructor deberá almacenar la fracción en forma reducida (es decir, la fracción 2/4 se almacenaría en el objeto como 1 en el numerador y 2 en el denominador).
- Incluir métodos de consulta y acceso de las variables miembro (getters y setters).
- Incluir métodos públicos para realizar las siguientes operaciones:
    - Multiplicación de dos números racionales
    - División de dos números racionales.

Para implementar los métodos multiplicación y división de números racionales de la clase Racional tener en cuenta que el resultado de la operación correspondiente se tiene que almacenar en un tipo Racional.



Para llevar a cabo estas tareas se requiere desarrollar un programa con las siguientes opciones:

1. Ingresar racionales.

Dar ingreso a dos números racionales, éstos serán los que intervengan en las operaciones hasta que se ingresen nuevos valores.

2 y 3. Operar multiplicación / división.

Con los valores almacenados en los objetos tipo Racional ingresados en la opción 1 se podrá realizar la operación seleccionada para luego mostrar su resultado. Observar, ¿si se eligió una de estas opciones, es posible realizarla si aún no se ejecutó la opción 1? ¿qué clase debería calcular y mostrar el resultado?

0. Salir. La ejecución cicla hasta que el usuario selecciona esta opción.

Enunciado 6

Se requiere gestionar la reservas de un cine, para esto es preciso desarrollar un proyecto en lenguaje Java que permita administrar la reserva de asientos en la sala de cine.

La aplicación debe manejar las clases Sala y Asiento. La clase Sala debe contener una cantidad fija de asientos y permitir su administración a través de un vector de asientos. Deben implementarse las relaciones de uso y de composición entre las clases, así como los principios de abstracción y encapsulamiento.

Descripción

1. Clase Sala:

- La clase Sala representa una sala de cine que tiene un número determinado de asientos.
- Contiene un vector de objetos Asiento para gestionar los asientos en la sala.
- Proporciona métodos para:
  * Reservar un asiento en una posición específica.
  * Cancelar la reserva de un asiento.
  * Consultar el estado de un asiento (si está reservado o disponible).
  * Mostrar el estado general de la sala, indicando qué asientos están ocupados y cuáles están libres.
- Aplicar encapsulamiento para que los atributos internos de la clase solo sean accesibles a través de métodos públicos.

2. Clase Asiento:

- La clase Asiento representa un asiento en la sala y contiene información sobre su disponibilidad.
- Cada Asiento tiene un identificador único dentro de la sala (un número) y un estado de reserva (reservado o libre).
- La relación de composición se establece ya que los asientos solo existen como parte de la sala.
- La relación de uso se puede aplicar cuando la Sala utiliza instancias de Asiento para la administración de reservas.

3. Requisitos de Abstracción y Encapsulamiento:

- La estructura interna de Sala y Asiento debe estar oculta para otras clases (encapsulamiento).
- Los métodos públicos en ambas clases deben ofrecer una interfaz clara para que otros componentes puedan interactuar con las funcionalidades sin conocer los detalles internos.

4. Restricciones:

- Se deben crear instancias de Asiento usando un vector en la clase Sala.

Ejemplo de funcionamiento:

1. Crear una instancia de Sala con 5 asientos.
2. Reservar el asiento en la posición 3.
3. Consultar el estado de cada asiento para ver cuáles están ocupados.
4. Cancelar la reserva del asiento en la posición 3.
5. Mostrar el estado de la sala.

Enunciado 7

Se requiere desarrollar una aplicación en Java que administre las plantas en un jardín botánico. La aplicación debe gestionar la relación entre una clase Jardín y una clase Planta.

La clase Jardín debe contener múltiples plantas y debe administrarlas utilizando un vector de plantas. La solución debe incluir las relaciones de uso y de composición entre las clases, así como los principios de abstracción y encapsulamiento.

Descripción:

1. Clase Jardín:

- La clase Jardín representa una colección de plantas en un espacio específico.

- Contiene un vector de objetos Planta que almacena las plantas presentes en el jardín.
- Proporciona métodos para:
  - Agregar una nueva planta al jardín en una posición específica.
  - Remover una planta de una posición determinada.
  - Consultar información sobre una planta específica dentro del jardín.
  - Mostrar un listado de todas las plantas en el jardín, incluyendo sus características (nombre, especie, estado de salud).
- Aplicar encapsulamiento para que los atributos internos de la clase solo sean accesibles mediante métodos públicos, permitiendo así el control sobre el acceso y la manipulación de los datos internos.

2. Clase Planta:

- La clase Planta representa una planta específica y contiene información sobre sus características.
- Cada Planta debe tener un identificador único dentro del jardín, un nombre, una especie y un estado de salud.
- La relación de composición se establece ya que las plantas solo existen como parte del jardín.
- La relación de uso se representa cuando el Jardín utiliza instancias de Planta para organizar y gestionar las especies presentes.

3. Requisitos de Abstracción y Encapsulamiento:

- La estructura interna de Jardín y Planta debe estar oculta, proporcionando una interfaz clara y sencilla para acceder a sus funcionalidades.
- Los métodos públicos de ambas clases deben permitir la interacción con las funcionalidades sin exponer detalles internos.

4. Restricciones:

- Se deben crear las instancias de Planta utilizando un vector en la clase Jardín.

Ejemplo de Funcionamiento:

1. Crear una instancia de Jardín con espacio para 5 plantas.
2. Agregar diferentes Planta al jardín, cada una con un nombre, una especie y un estado de salud.
3. Consultar la información de una planta específica.
4. Remover una planta del jardín.
5. Mostrar el listado general de todas las plantas en el jardín, junto con sus características.

Enunciado 8

Se requiere desarrollar una aplicación en Java para gestionar el proceso de una elección democrática. La aplicación debe administrar las relaciones entre una clase Elección y una clase Candidato. La Elección debe contener una lista de candidatos que participarán en el proceso electoral, almacenada en un vector. La solución debe implementar las relaciones de uso y composición entre las clases, aplicando los principios de abstracción y encapsulamiento. No se permite el uso de polimorfismo ni herencia.

Descripción

1. Clase Elección:

- La clase Elección representa el proceso electoral y contiene la lista de candidatos.

- Utiliza un vector de objetos Candidato para almacenar los candidatos participantes en la elección.
- Proporciona métodos para:
  - Inscribir un candidato en la elección.
  - Eliminar un candidato de la elección.
  - Registrar un voto para un candidato.
  - Consultar el número total de votos recibidos por cada candidato.
  - Mostrar la lista de todos los candidatos con sus respectivos conteos de votos.
- Aplicar encapsulamiento para que los atributos internos de la clase solo sean accesibles a través de métodos públicos, manteniendo el control sobre el acceso y la manipulación de los datos internos.

2. Clase Candidato:

- La clase Candidato representa a un candidato en la elección y almacena su información.
- Cada Candidato tiene un identificador único, el nombre del candidato y el número de votos recibidos.
- La relación de composición se establece ya que los candidatos solo existen como parte de la Elección.
- La relación de uso se manifiesta cuando la Elección utiliza instancias de Candidato para organizar y gestionar los candidatos participantes y su información de votos.

3. Requisitos de Abstracción y Encapsulamiento:

- La estructura interna de Elección y Candidato debe estar oculta, de forma que solo sea accesible mediante métodos públicos y proporcionando una interfaz clara y sencilla para interactuar con la funcionalidad de cada clase.
- Los métodos públicos deben permitir la funcionalidad requerida sin exponer detalles internos de la implementación.

4. Restricciones:

- Se deben crear las instancias de Candidato utilizando un vector en la clase Elección.

<u>Ejemplo de Funcionamiento:</u>

1. Crear una instancia de Elección para una elección con 3 candidatos.
2. Inscribir a distintos Candidato en la elección, cada uno con un nombre y un identificador.
3. Registrar votos para los candidatos en la elección.
4. Consultar el total de votos de un candidato específico.
5. Eliminar un candidato de la elección.
6. Mostrar el listado general de candidatos y los votos recibidos por cada uno.