ITID433
Programación Avanzada
Facultad de Ingeniería Eléctrica y Electrónica

Práctica 03

Abstracción

Objetivos:

Esta tercera práctica de la asignatura persigue los siguientes objetivos:

- Practicar la creación de clases.
- Reforzar la comprensión sobre objetos, estado y comportamiento.

Instrucciones y Actividades:

1. Marco Teórico

Clase

Es una plantilla o un esquemático que define las características y comportamientos comunes de los objetos de su tipo.

Elementos: Los elementos de una clase son:

- Atributos: Variables que representan las características o propiedades de la clase.
- Métodos: Funciones que definen los comportamientos o acciones que los objetos de la clase pueden realizar.
- Constructores: Son como métodos cuyo objetivo es inicializar objetos de una clase.

Ejemplo:

```
public class Carro {
    String fabricante;
    String modelo;
    int año;
    public void arrancarMotor() {
        System.out.println("Motor iniciado");
    }
}
```

Objeto

Un objeto es una instancia de una clase. Representa una entidad concreta con un estado y un comportamiento definidos por su clase.

El estado de un objeto se define por los valores de sus atributos, mientras que su comportamiento se define por los métodos que puede ejecutar.

Ejemplo:

```
Carro miCarro = new Carro();
```

Estado

El estado de un objeto se refiere a los valores actuales de sus atributos.

Ejemplo:

```
Carro miCarro = new Carro();
// El estado del objeto miCarro:
// fabricante = null,
// modelo = null,
// año = 0
// Dado que no hay constructor,
// el estado fue inicializado por Java
```

Comportamiento

El comportamiento de un objeto se refiere a las acciones que puede realizar, definidas por los métodos de su clase.

Ejemplo:

```
miCarro.arrancarMotor();
```

Constructor sin parámetros

Un constructor sin parámetros que inicializa los atributos con valores predeterminados.

Ejemplo:

```
public Carro() {
    this.fabricante = "Desconocido";
    this.modelo = "Desconocido";
    this.año = 0;
}
```

Constructor con parámetros

Un constructor que toma argumentos para inicializar los atributos del objeto con valores específicos.

Ejemplo:

```
public Carro(String fabricante, String modelo, int año) {
    this.fabricante = fabricante;
```

```
        this.modelo = modelo;
        this.año = año;
    }
```

Constructor por copia

Un constructor que crea un nuevo objeto copiando los valores de los atributos de otro objeto de la misma clase.

Ejemplo:

```
    public Carro(Carro otroCarro) {
        this.fabricante = otroCarro.fabricante;
        this.modelo = otroCarro.modelo;
        this.año = otroCarro.año;
    }
```

Getters y Setters

Son métodos que permiten acceder y modificar los valores de los atributos privados de una clase.

- Getters: Métodos que devuelven el valor de un atributo.
- Setters: Métodos que establecen el valor de un atributo.

Ejemplo:

```
    public String getFabricante() {
        return fabricante;
    }
    public void setFabricante(String fabricante) {
        this.fabricante = fabricante;
    }
```

Variables de Clase

Son variables que son compartidas por todas las instancias de una clase.

Ejemplo:

```
    static int cantCarros = 0;
```

Métodos de Clase

Son métodos que operan a nivel de clase, no a nivel de instancia.

Ejemplo:

```
    public static int obtenerNumeroCarros() {
        return cantCarros;
```

```
}
```

2. Clase Carro

1. Crea una clase llamado `Carro`.
   En la clase, con base en el esquema que se indica a continuación completa las secciones según los pasos indicados a continuación.

```
public class Carro {
     // variables de clase
     // variables de instancia
     // Constructores
     // métodos de clase
     // métodos de instancia
}
```

2. Agrega el código que permita almacenar, a nivel de clase, el número de autos que se creen de la clase `Carro`. Se debe contar con una variable de tipo `int` que sea `static`, y que esté iniciada en `0`.
   Agrega en la sección variables de clase:

```
static int numCarros = 0;
```

3. Agrega el código que permita almacenar, a nivel de clase, el nombre de la fábrica en la que se almacenarán los carros. Se debe contar con una variable de tipo `String` que sea `final static`, y que esté iniciada con las iniciales de sus nombres (por ejemplo, en mi caso DM).
   Agrega en la sección variables de clase:

```
final static String nombreFabrica = "DM";
```

4. Agrega el código que permita almacenar, a nivel de instancia, el fabricante, modelo y año de fabricación de un carro. Se debe contar con variables de tipo `String` para el fabricante y modelo, y de tipo `int` para el año de fabricación.
   Agrega en la sección variables de instancia:

```
String fabricante;
String modelo;
int año;
```

5. Agrega un constructor sin parámetros que permita inicializar el estado de un objeto `Carro`, considera que al inicializar el fabricante y modelo debe ser "`Desconocido`" mientras que para el año usa `0`. Para llevar el control de Carros creados, realiza el incremento del valor de `numCarros`.
   Agrega en la sección constructor:

```
public Carro() {
     this.fabricante = "Desconocido";
     this.modelo = "Desconocido";
     this.año = 0;
     numCarros++;
}
```

6. Agrega un constructor con parámetros que permita inicializar el estado de un objeto `Carro`. El estado se inicializa con los parámetros que se pasen al constructor como argumentos. Para llevar el control de Carros creados, realiza el incremento del valor de `numCarros`.
   Agrega en la sección constructor:

```
public Carro(String fabricante, String modelo, int
año) {
     this.fabricante = fabricante;
     this.modelo = modelo;
     this.año = año;
     numCarros++;
}
```

7. Agrega un constructor por copia que permita inicializar el estado de un objeto `Carro` copiando la información desde otro objeto del mismo tipo. Para llevar el control de Carros creados, realiza el incremento del valor de `numCarros`.
   Agrega en la sección constructor:

```
public Carro(Carro otro) {
     this.fabricante = otro.fabricante;
     this.modelo = otro.modelo;
     this.año = otro.año;
     numCarros++;
}
```

8. Agrega 2 *getters* y 2 *setteres* para controlar el acceso al estado del objeto. Los *setters* deben devolver un `boolean`, siendo `true` si todo fue exitoso o `false` si hubo un problema. Asegúrate que, en el año, los valores sean positivos y superiores a 1900. También asegúrate que, en fabricante el usuario no vaya a dejarlos vacíos.
   Agrega en la sección métodos de instancia:

```
public String getFabricante() {
     return fabricante;
}
public boolean setFabricante(String fabricante) {
     if(fabricante==null || fabricante.isEmpty())
     {
```

```
            return false;
        }
        this.fabricante = fabricante;
        return true;
    }

    public int getAño() {
        return año;
    }
    public boolean setAño(int año) {
        if(año <= 1900)
            return false;
        this.año = año;
        return true;
    }
```

9. Agrega 1 *getter* y 1 *settere1* para controlar el acceso al modelo y asegúrate que el usuario no vaya a ingresar un valor vacío.

10. Agrega el método `arrancar`. El método `arrancar` debe retornar un `String` indicando que el carro ha arrancado.
    Agrega en la sección métodos de instancia:
    ```
    public String arrancar() {
        return "El auto ha arrancado";
    }
    ```

11. Agrega el método `desplegarInformación`. El método debe retornar un `String` con el estado del carro.
    Agrega en la sección métodos de instancia:
    ```
    public String desplegarInformación() {
        String msj = "Auto fabricado por: " +
    fabricante;
        msj += " modelo: " + modelo;
        msj += " fabricado en el año: " + año;
        return msj;
    }
    ```

12. Agrega el método estático `presentarNumAutos`. El método debe retornar un `int` que contenga el número de autos creados en la fábrica.
    Agrega en la sección métodos de clase:
    ```
    public static int presentarNumAutos() {
        return numCarros;
    }
    ```

13. Agrega el método estático `reiniciarNumAutos`. El método debe reiniciar el contador de autos.

14. Agrega la clase y `Main`, y en la misma agrega el método `main` para realizar pruebas de la clase `Carro`.
    Agrega en `Main.java`:

```
public class Main {
    public static void main(String[] args) {
     // Presenta información de la fabrica
     // y la cuenta
     System.out.println("Fábrica:          "          +
     Carro.nombreFabrica);
     System.out.println("En   la   fábrica   se   han
     producido: " + Carro.numCarros);

     // Crea un objeto Carro usando el constructor sin
     // parámetros
     Carro c1 = new Carro();

     // Presenta la info de c1
     System.out.println(c1.desplegarInformación());

     // Presenta la cant de autos de la fabrica
     System.out.println();
     System.out.println("En   la   fábrica   se   han
     producido: " + Carro.numCarros);

     // Intenta actualizar el estado de c1
     // en resultados indica que campos se cambiaron y
     // cuales no
     if (c1.setFabricante("Toyota")) {
         System.out.println("Fabricante ingresado");
     } else {
         System.out.println("ERROR");
     }

     if (c1.setModelo("")) {
         System.out.println("Modelo ingresado");
     } else {
         System.out.println("ERROR");
     }

     if (c1.setAño(-1)) {
         System.out.println("Año ingresado");
     } else {
         System.out.println("ERROR");
     }

     // Presenta la info de c1
```

```
        System.out.println(c1.desplegarInformación());
        // actualiza la info de c1 con datos de un carro
        // Presenta la info de c1

        // Presenta la cant de autos de la fabrica
        System.out.println();
        System.out.println("En la fábrica se han
        producido: " + Carro.numCarros);

        // Crea un objeto Carro usando el constructor con
        // parámetros (indica qué línea funciona, y por
        // qué funciona o no – esto debe ir en la sección
        // resultados), en caso de que no funcione corrige
        // el problema y presenta la info del
        // carro, al final presenta la cant de carros
        // (indica en el informe si el número obtenido es
        // coherente con el número de objetos creados.
        Carro c2 = new Carro("Suzuki");
        System.out.println(c2.desplegarInformación());
        Carro c3 = new Carro("Suzuki", "");
        System.out.println(c3.desplegarInformación());
        Carro c4 = new Carro("Suzuki", "", -1);
        System.out.println(c4.desplegarInformación());
        Carro c5 = new Carro(null, null, 0);
        System.out.println(c5.desplegarInformación());
        Carro c6 = new Carro("Suzuki", "Scross", 2000);
        System.out.println(c6.desplegarInformación());
        System.out.println();
        System.out.println("En la fábrica se han
        producido: " + Carro.numCarros);

        // Crea un objeto Carro usando el constructor por
        // copia usando c1, y despliega la info de la
        // copia. Indica el número de carros producidos.
        // analiza si el número es cohente con los objetos
        // creados hasta el momento
        Carro cCopia = new Carro(c1);
        System.out.println(cCopia.desplegarInformación
());
        System.out.println();
        System.out.println("En la fábrica se han
producido: " + Carro.numCarros);

        // Crea una referencia de tipo Carro y asígnala
        // al carro copiado despliega la info de la
        // referencia. Indica el número de carros
        // producidos.
        // analiza si el número es coherente con los
        // objetos creados hasta el momento
        Carro carroRef = cCopia;
```

```
System.out.println(carroRef.desplegarInformación
());

      System.out.println();
      System.out.println("En  la  fábrica  se  han
      producido: " + Carro.numCarros);

      // Actualiza la info de carroRef
      // fabricante: Peugeot
      // modelo: 306
      // año: 1990
      // ahora compara la info de carroRef con cCopia
      // y con c1. Explica en el informe porque
      // la información es igual o si difiere
    }
}
```

3. Clase Carta

1. Crea una clase llamado `Carta`.
2. Como atributos define la familia ("Trébol", "Corazones", "Picas" y "Diamantes") y la cara de la carta ("As", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q" y "K"), ambos atributos deben ser de tipo `String`.
3. Agrega un tercer atributo que sea de tipo `boolean` y que permita establecer el valor de la carta con la cara "As" (si es `true` "As" valdrá 11, si es `false` "As" valdrá 1).
4. Define un constructor con parámetros que acepte la familia y la cara, además el constructor en el caso de que la carta sea "As" establecerá el valor del atributo de tipo `boolean` en `true`.
5. Define dos *getters*, uno para obtener la cara de la carta y el otro la familia de la carta.
6. Define un método `valorCarta` que permita obtener el valor de la carta. Este método devolverá un `int` cuyo valor dependa del rango (si el rango es As debe devolver 11 o 1, si el rango es 2 devuelve 2, …, si el rango es J devuelve 10, si el rango es Q devuelve 10, y si el rango es K devuelve 10. Se sugiere usar la sentencia `switch`).
7. Define un método `toString` que devuelva un `String` que concatene la cara de la carta, la palabra de y la familia, por ejemplo "As de Corazones".
8. Define un setter que permita cambiar la variable de tipo `boolean` para controlar el valor del As.

```
public class Cart {
  String cara;
  String familia;
  boolean esAsAlto;
```

```
  public Carta(String cara, String familia) {
        this.cara = cara;
        this.familia = familia;
        if(cara.equals("As")
             esAsAlto = true;
  }

  public int valor() {
        // completar el código
  }

  public void setAsAlto(boolean esAlto) {
        this.esAsAlto = esAlto;
  }

  public String getCara() {
        return cara;
  }

  public String getFamilia() {
         return familia;
  }

  public String toString() {
         return cara + " de " + familia;
  }
}
```

9. Agrega la clase `Main` con el método `main` para realizar las pruebas de tu clase.
10. Crea al menos 5 instancias de cartas (As de Corazones, J de Trébol, Q de Diamante, 3 de Picas y As de Trébol).
11. Modifica usando el *setter* el valor del "As de Trébol" para que sea 1.
12. Utiliza el método `toString` para presentar la información de cada carta.
13. Utiliza los *getters* para presentar la cara y familia de "J de Trébol".
14. Realiza las llamadas de los métodos `valor` para conocer el valor de cada carta.
15. Considera que las 5 instancias corresponden a la mano de un jugador, presenta el resultado de la suma de sus valores.
16. Los resultados de las pruebas deben ser colocados en la sección resultados.

4. Entregables

Para esta práctica debes realizar lo siguiente:

1. Completa el código solicitado.
2. Ubica la carpeta del proyecto generada, en caso de que hayas usado Eclipse o IntelliJ, elimina la carpeta `bin` o `out` y procede a comprimir la carpeta del proyecto. Esta carpeta comprimida (.ZIP), con el nombre cumpliendo el formato indicado en clase, es la que debes subir en el Práctica 03. En caso de que uses Replit, descarga solamente el archivo `.java`, y ese archivo comprímelo y cárgalo en la plataforma.

5. Realizado por:

David Mejía N.