Actividades de apropiación

Programación Orientada a Objetos

Diagrama de clases

Observa el siguiente video,

https://www.youtube.com/watch?v=_r1n_ipED8s

- ✓ Contesta,

¿Qué es un diagrama de clases?

¿Por qué son importantes los diagramas de clases en un desarrollo de software?

Relaciones entre clases

Observa el siguiente video,

https://www.youtube.com/watch?v=kpSP1W7T5VQ

Visita la página,

https://blog.visual-paradigm.com/es/what-are-the-six-types-of-relationships-in-uml-class-diagrams/

- ✓ Consulta y contesta,

Herencia:

¿Qué es la herencia en la programación orientada a objetos?

¿Cuál es la diferencia entre una clase base (superclase) y una clase derivada (subclase)?

¿Cómo se implementa la herencia en tu lenguaje de programación preferido?

¿Qué es la sobrecarga de métodos y cómo se relaciona con la herencia?, ¿Lo permite Python?

¿Qué ventajas y desventajas tiene el uso de la herencia?

Programación orientada a objetos - Actividad No 3

Composición:

¿Qué es la composición en la programación orientada a objetos?

¿Cómo difiere la composición de la herencia?

¿Puedes proporcionar un ejemplo donde la composición sea más adecuada que la herencia?

¿Qué se entiende por " es parte de" o "posee un" en el contexto de la composición?

¿Cómo se puede implementar la composición en un lenguaje de programación?

Agregación:

¿Qué es la agregación en la programación orientada a objetos?

¿Cómo difiere la agregación de la composición?

¿Puedes proporcionar un ejemplo donde la agregación sea más adecuada que la composición?

¿Qué se entiende por "tiene un" o "contiene un" en el contexto de la agregación?

¿Cómo se puede implementar la agregación en un lenguaje de programación?

Dependencia:

¿Qué es una relación de dependencia entre clases?

¿Por qué es importante minimizar las dependencias entre clases?

¿Qué técnicas puedes usar para reducir las dependencias entre clases en tu código?

Interfaces y Polimorfismo:

Programación orientada a objetos - Actividad No 3

¿Qué es una interfaz y cómo se utiliza en la programación orientada a objetos?

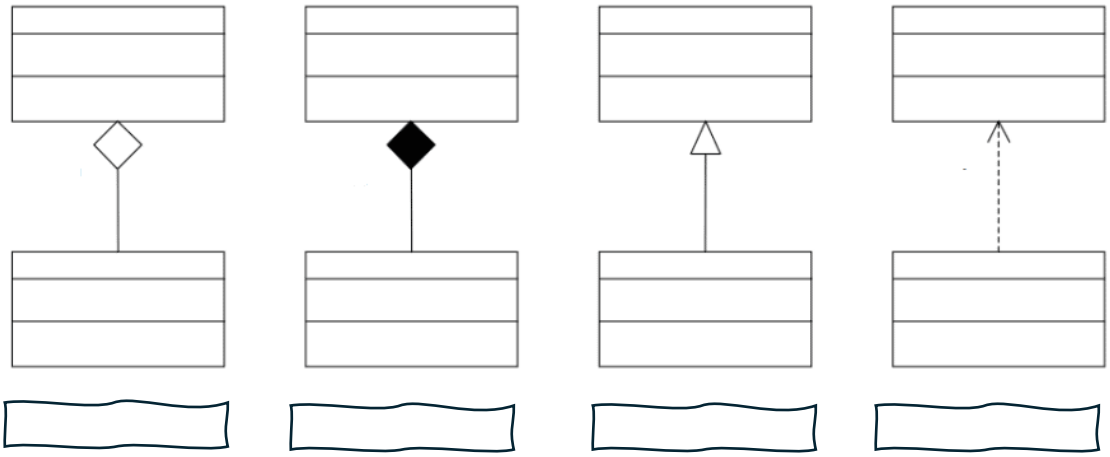
¿Cómo se relacionan las interfaces con el polimorfismo?

¿Qué es el polimorfismo y cómo se implementa?

¿Cuáles son los beneficios de usar interfaces en lugar de clases concretas?

Diagramación

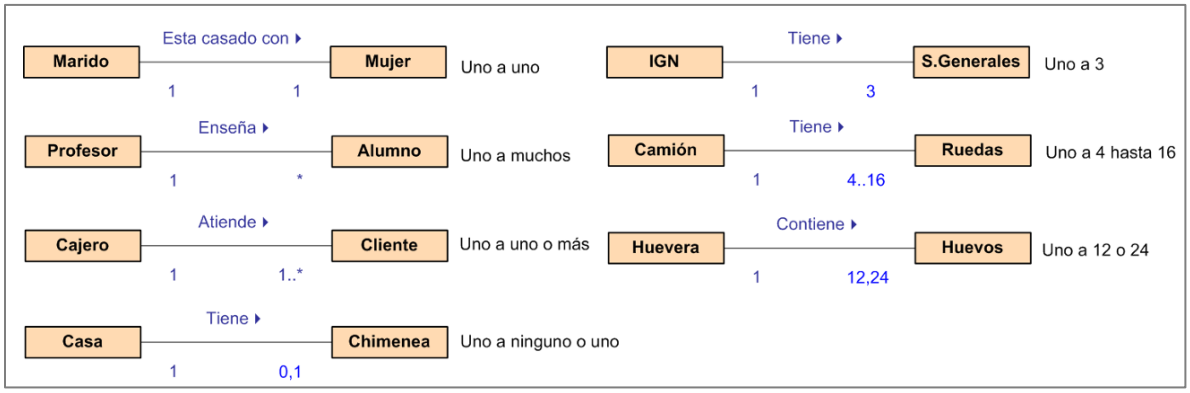
- ✓ Observa la gráfica a continuación y escribe el nombre correspondiente de la relación entre las dos clases.



Multiplicidad de las clases

La multiplicidad indica cuántas instancias de una clase pueden estar asociadas con una instancia de otra clase. En otras palabras, describe la cantidad de objetos que pueden participar en una relación.

Ejemplos de Multiplicidad



Tomado de https://www.gbif.es/wp-content/uploads/2018/01/Modelos-de-datos-Lenguaje-UML-Paloma-Abad-y-Alejandra-Sanchez.pdf

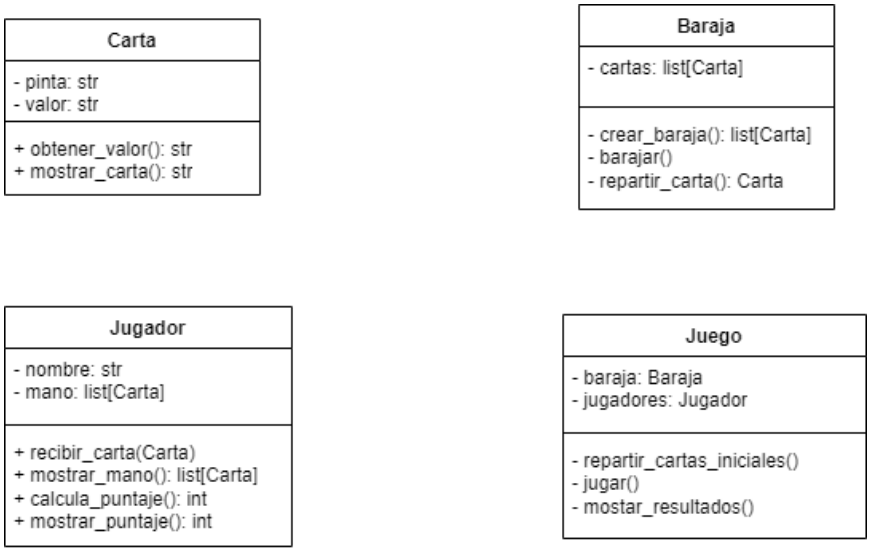
Ejercicio práctico - Blackjack

El blackjack, también llamado veintiuno, es un juego de cartas, propio de los casinos con una o más barajas inglesas de 52 cartas sin los comodines, que consiste en sumar un valor lo más próximo a 21 pero sin pasarse. En un casino cada jugador de la mesa juega únicamente contra el crupier, intentando conseguir una mejor jugada que este. El crupier está sujeto a reglas fijas que le impiden tomar decisiones sobre el juego. Por ejemplo, está obligado a pedir carta siempre que su puntuación sume 16 o menos, y obligado a plantarse si suma 17 o más. Las cartas numéricas suman su valor, las figuras suman 10 y el as vale 11 o 1, a elección del jugador. En el caso del crupier, los ases valen 11 mientras no se pase de 21, y 1 en caso contrario. La mejor jugada es conseguir 21 con solo dos cartas, esto es con un As más carta de valor 10. Esta jugada se conoce como Blackjack o 21 natural. Un blackjack gana sobre un 21 conseguido con más de dos cartas. (“Blackjack,” n.d.)

Observa el siguiente video,

https://www.youtube.com/watch?v=ecKMGu996z4

- ✓ A continuación, se presentan las clases necesarias para representar el juego de Blackjack. Sin embargo, es necesario relacionar estas clases entre sí para que tengan sentido en el desarrollo del juego. Utilizando los conocimientos y actividades previas, completa el diagrama de clases de manera que represente adecuadamente el juego de cartas.



✓ **Desarrollo en Python**

Utilizando el lenguaje de programación Python y basándonos en el diagrama de clases resultante anteriormente, construye un programa que simule el juego de Blackjack.

Existen convenciones de escritura de nombres para variables, métodos, funciones, clases y constantes en diferentes lenguajes de programación, estas son reglas establecidas para mejorar la legibilidad, coherencia y mantenimiento del código. Aquí te explico las convenciones más comunes para variables, métodos, funciones, clases y constantes en diferentes lenguajes de programación.

Python

1. Variables y funciones: snake_case

Todas las letras en minúsculas.
Las palabras se separan con guiones bajos (_).
Ejemplos: mi_variable, calcular_suma.

2. Clases: UpperCamelCase (Pascal Case)

La primera letra de cada palabra es mayúscula.
Ejemplos: MiClase, ClaseEjemplo.

3. Métodos: snake_case

Similar a las variables y funciones.
Ejemplos: crear_baraja, obtener_datos.

4. Constantes: UPPER_CASE

Todas las letras en mayúsculas.
Las palabras se separan con guiones bajos.
Ejemplos: PI, MAXIMO_VALOR.

Programación orientada a objetos - Actividad No 3

Java

1. Variables y métodos: lowerCamelCase

La primera palabra comienza en minúscula, las palabras subsiguientes empiezan con mayúscula.

Ejemplos: miVariable, calcularSuma.

2. Clases e interfaces: UpperCamelCase (Pascal Case)

Cada palabra empieza con mayúscula.

Ejemplos: MiClase, ClaseEjemplo.

3. Constantes: UPPER_CASE

Todas las letras en mayúsculas.

Las palabras se separan con guiones bajos.

Ejemplos: PI, MAXIMO_VALOR.

JavaScript

1. Variables y funciones: lowerCamelCase

Similar a Java.

Ejemplos: miVariable, calcularSuma.

2. Clases: UpperCamelCase (Pascal Case)

Similar a Java.

Ejemplos: MiClase, ClaseEjemplo.

3. Constantes: UPPER_CASE

Similar a Python y Java.

Ejemplos: PI, MAXIMO_VALOR.

C#

1. Variables y métodos: lowerCamelCase

Similar a Java.

Ejemplos: miVariable, calcularSuma.

2. Clases y métodos públicos: UpperCamelCase (Pascal Case)

Similar a Java.

Ejemplos: MiClase, ClaseEjemplo.

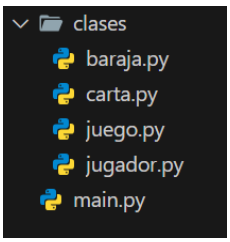
3. Constantes: UPPER_CASE

Similar a otros lenguajes.

Ejemplos: PI, MAXIMO_VALOR.

Explicación código:

Estructura:



Clase Carta

```python
class Carta:
    def __init__(self, pinta, valor):
        self.__pinta = pinta
        self.__valor = valor

    def obtener_valor(self):
        return self.__valor

    def mostrar_carta(self):
        return f"|{self.__valor}{self.__pinta}|"

```

Clase Baraja

```python
import random
from clases.carta import Carta

class Baraja:


    def __init__(self):
        self.__cartas = self.__crear_baraja()
        self.__barajar()

    def __crear_baraja(self):
        pintas = ['♥', '♦', '♣', '♠']
        valores = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
        cartas = [Carta(pinta, valor) for pinta in pintas for valor in valores]
        return cartas

    def __barajar(self):
        random.shuffle(self.__cartas)

    def repartir_carta(self):
        return self.__cartas.pop() if self.__cartas else None

```

```
pintas = ['♥', '♦', '♣', '♠']
valores = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
```

```
cartas = [Carta(pinta, valor) for pinta in pintas for valor in valores]
```

Este código utiliza una **comprensión de listas** para crear una lista de objetos Carta.

Comprensión de listas: Es una forma concisa de crear listas en Python.

En la creación de esta lista tiene dos bucles for anidados.

Carta (pinta, valor): Esta es la expresión que se evalúa para cada combinación de pinta y valor. Carta es una clase y pinta y valor son los parámetros del constructor de la clase Carta.

for pinta in pintas: Este es el primer bucle for. Recorre todos los elementos en pintas. pintas es una lista que contiene diferentes pintas (Corazones, diamantes, tréboles, picas).

for valor in valores: Este es el segundo bucle for. Para cada pinta, este bucle recorre todos los elementos en valores. valores es una lista que contiene diferentes valores (2, 3, ..., J, Q, K, A).

cartas: Finalmente, la lista resultante de la comprensión de listas se asigna a la variable cartas.

Clase Jugador

```
class Jugador:
    def __init__(self, nombre):
        self.__nombre = nombre
        self.__mano = []

    def recibir_carta(self, carta):
        self.__mano.append(carta)

    def mostrar_mano(self):
        return [carta.mostrar_carta() for carta in self.__mano]

    def calcular_puntaje(self):
        puntaje = 0
        ases = 0
        valores = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10, 'J': 10, 'Q': 10, 'K': 10, 'A': 11}
        for carta in self.__mano:
            puntaje += valores[carta.obtener_valor()]
            if carta.obtener_valor() == 'A':
                ases += 1
        while puntaje > 21 and ases:
            puntaje -= 10
            ases -= 1
        return puntaje

    def mostrar_puntaje(self):
        return self.calcular_puntaje()

```

Clase Juego

```python
from clases.baraja import Baraja
from clases.jugador import Jugador
class Juego:
    def __init__(self, nombres_jugadores):
        self.__baraja = Baraja()
        self.__jugadores = [Jugador(nombre) for nombre in nombres_jugadores]
        self.__repartir_cartas_iniciales()

    def __repartir_cartas_iniciales(self):
        for _ in range(2):
            for jugador in self.__jugadores:
                jugador.recibir_carta(self.__baraja.repartir_carta())

    def jugar(self):
        for jugador in self.__jugadores:
            print(f"Turno de {jugador._Jugador__nombre}")
            while True:
                print(f"Mano: {jugador.mostrar_mano()} (Puntaje: {jugador.mostrar_puntaje()})")
                accion = input("¿Quieres otra carta? (s/n): ")
                if accion.lower() == 's':
                    jugador.recibir_carta(self.__baraja.repartir_carta())
                    if jugador.mostrar_puntaje() > 21:
                        print("¡Te has pasado de 21!")
                        break
                else:
                    break

        self.__mostrar_resultados()

    def __mostrar_resultados(self):
        for jugador in self.__jugadores:
            print(f"{jugador._Jugador__nombre}: {jugador.mostrar_mano()} (Puntaje: {jugador.mostrar_puntaje()})")
        ganador = max(self.__jugadores, key=lambda jugador: jugador.mostrar_puntaje() if jugador.mostrar_puntaje() <= 21 else 0)
        print(f"El ganador es {ganador._Jugador__nombre} con un puntaje de {ganador.mostrar_puntaje()}")

```

main

```python
from clases.juego import Juego
if __name__ == "__main__":
    nombres_jugadores = ["Jugador 1", "Jugador 2"]
    juego = Juego(nombres_jugadores)
    juego.jugar()
```

✓ **Desarrollo en Python**

Después de haber desarrollado el código para el juego, es importante que expliques detalladamente las funciones de Python que se han utilizado. Asegúrate de cubrir los siguientes aspectos:

Descripción de las funciones: Proporciona una explicación clara y concisa de cada función utilizada, indicando su propósito y cómo contribuye al funcionamiento del juego.

Parámetros y retorno: Detalla los parámetros que recibe cada función y lo que devuelve. Esto ayudará a entender mejor cómo interactúan las funciones entre sí y con el resto del código.

Documentación y comentarios: Asegúrate de que tu código esté bien documentado con comentarios claros que expliquen el flujo de lógica y las decisiones tomadas durante el desarrollo.

Bibliografía

Wikipedia. (n.d.). Blackjack. En Wikipedia, la enciclopedia libre. Recuperado el [16-07-2024], de https://es.wikipedia.org/wiki/Blackjack

Otero V. Mari C., A. A. (s.f.). Capítulo 3: diagramas de clases. Tomado de, http://www.vc.ehu.es/jiwotvim/IngenieriaSoftware/Teoria/BloqueII/UML-3.pdf