Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-2

Midterm

02 de Octubre 2025

Instrucciones

- La evaluación consta de dos partes: 20 preguntas de selección múltiple y 2 preguntas de desarrollo.
- Cada una de las secciones (alternativas y desarrollo) son independientes y valen un 50 % de la nota final. Para calcular la nota de cada sección, se considera que todas las preguntas de alternativas valen lo mismo, mientras que la sección de desarrollo indica cuánto vale cada pregunta.
- Recibirás una hoja de respuestas que deberás rellenar con tus datos y las respuestas correspondientes a cada alternativa o pregunta de desarrollo. Sólo se corregirá la hoja de respuestas. **Ten mucha precaución de anotar correctamente tus datos**.
- Cada pregunta de selección múltiple tiene únicamente 1 alternativa correcta. Responder con 2 o más alternativas implicará dejar inválida esa pregunta y se considerará incorrecta. Cada pregunta presenta el mismo puntaje y no se descontará por respuesta incorrecta.
- Debes completar con tus datos personales en cada hoja utilizada. Luego debes entregar de vuelta **todas** las hojas de respuesta (tanto para alternativas como de desarrollo) de la prueba, aunque estas se encuentren en blanco.
- Solo podrás retirarte de la sala una vez hayan transcurrido 45 minutos desde que inició de la evaluación.
- Durante la evaluación se realizarán 2 rondas de preguntas. Estas preguntas únicamente serán respondidas por los profesores del ramo.
- En caso de que sea necesario, podrás solicitar hojas blancas para apoyar al desarrollo de la prueba. Para esto levanta la mano y espera que un ayudante se acerque a tu puesto.

Selección múltiple

1. En Git, ¿qué contiene un *commit*?

   A) Solo los archivos modificados hasta ese momento.

   B) Solo los archivos nuevos agregados hasta ese momento.

   C) El estado completo del proyecto hasta ese momento.

   D) Solo los archivos eliminados hasta ese momento.

   E) La configuración del repositorio en ese momento.

2. Según `PEP8`, ¿cuál es la convención correcta para nombrar clases y métodos en Python?

   A) Clase en `snake_case`, métodos en `CamelCase`.

   B) Clase en `UPPERCASE`, métodos en `lowercase`.

   C) Clase en `CamelCase`, métodos en `snake_case`.

   D) Clase en `snake_case`, métodos en `snake_case`.

   E) Clase en `CamelCase`, métodos en `CamelCase`.

3. Un estudiante desea ejecutar *git push* para subir sus cambios al repositorio remoto. Considerando que los archivos modificados todavía no están en el *staging area*, ¿cuál o cuáles de las siguientes condiciones son necesarias para que el *push* se realice exitosamente?

   I. Estar conectado a alguna red que permita acceder al repositorio remoto.

   II. Tener el repositorio local sincronizado con todos los cambios del repositorio remoto.

   III. Solo hacer *git commit* a los archivos modificados antes de ejecutar *git push*.

   A) Solo I.

   B) I y II.

   C) II y III.

   D) I, II y III.

   E) Ninguna es necesaria.

4. Considera la siguiente estructura de archivos de un proyecto y el contenido del archivo `main.py`:

```
mi_proyecto/
├── src/
│   ├── procesamiento/
│   │   └── datos.py
│   └── main.py
└── data/
    └── archivo.csv
```

```python
from src.procesamiento.datos import procesar_archivo

if __name__ == "__main__":
    ruta_archivo = "data/archivo.csv"
    resultado = procesar_archivo(ruta_archivo)
    print(resultado)
```

Asumiendo que cuentas con un intérprete de Python 3 accesible con el comando `python3` y que el directorio actual es `mi_proyecto/src`, ¿cuál de las siguientes alternativas es **falsa**?

A) Al ejecutar el archivo `main.py`, este funcionará porque está en el directorio actual y se logra acceder al archivo utilizando una ruta relativa.

B) La importación "`from src.procesamiento.datos import procesar_archivo`" fallará porque no se puede encontrar el módulo "`src`" desde el directorio actual.

C) Si cambias la ruta a "`os.path.join("data", "archivo.csv")`", el código será más portable entre diferentes sistemas operativos.

D) Si ejecutas el código usando "`python3 ../main.py`" causará un error.

E) Usar "`pathlib.Path("data/archivo.csv")`" en lugar de un *string* es una alternativa más flexible para manejar rutas.

5. Selecciona la estructura de datos que cumpla todas las propiedades que se presentan a continuación:

I. Mutable

II. *Hashable*

III. Indexable por posición

IV. No presenta un orden secuencial

A) Lista

B) Tupla

C) Set

D) Diccionario

E) Ninguna de las anteriores

6. Respecto a `*args` y `**kwargs`, indica cuál o cuáles de las siguientes afirmaciones son verdaderas:

   I. Pueden presentar valores por defecto.

   II. Pueden ubicarse en cualquier posición entre los argumentos de una función.

   III. Pueden estar vacíos.

   A) Solo I

   B) Solo III

   C) I y II

   D) II y III

   E) I, II y III

7. Se tiene una estructura de datos vacía a la cual se agregan valores numéricos y luego se calcula el promedio de ellos, sin embargo el resultado no es el correcto. ¿A qué puede deberse esto?

   I. La estructura guarda la información mediante el uso de *hash*.

   II. La estructura es *hasheable* y secuencial.

   III. La estructura es *hasheable* y no secuencial.

   A) Solo I

   B) Solo II

   C) Solo III

   D) I y II

   E) I y III

8. Suponiendo que quieres modelar el movimiento entre carpetas en un sistema de archivos usas:

   - "`cd Carpeta`" para ingresar a una carpeta hija llamada "`Carpeta`".
   - "`cd ..`" para retroceder a la carpeta padre.

   ¿Qué estructura de datos es más adecuada para modelar el problema?

   A) Tupla

   B) Stack

   C) Cola

   D) Diccionario

   E) Set

9. ¿Cuál de las siguientes alternativas es **falsa** respecto al concepto de herencia en Python?

   A) Es posible heredar de *built-ins*, como listas o tuplas.

   B) Gracias a la herencia es posible implementar el concepto de *overriding*.

   C) Gracias a la herencia es posible implementar el concepto de *overloading*.

   D) No hay límite respecto al número de clases de las cuales una subclase puede heredar.

   E) Una superclase puede a su vez ser subclase de otra superclase.

10. ¿Cuál de las siguientes alternativas es **verdadera** sobre las clases abstractas?

    A) Una clase abstracta solo puede heredar de una única superclase a la vez, que es la superclase `ABC`.

    B) No es posible que una subclase herede de dos clases abstractas a la vez, solo puede heredar de una.

    C) Todos los métodos de una clase abstracta deben tener el decorador `@abstractmethod` o el código generará una excepción.

    D) Si una clase abstracta define un `__init__` con atributos, estos deben ser sobreescritos en las subclases o el código generará una excepción.

    E) Si una subclase hereda de una clase abstracta y no sobreescribe todos sus métodos abstractos, también es considerada como una clase abstracta.

11. Respecto a Iterables e Iteradores, asocie cada una de las siguientes afirmaciones con el concepto que corresponda:

    I. Tiene un estado interno de iteración.

    II. Una vez consumido no puede ser reutilizado.

    III. Necesita de la implementación del método `__next__`.

    IV. Implementa el método `__iter__`.

A)	Iterable: I y II	Iterador: III y IV
B)	Iterable: I, II y III	Iterador: IV
C)	Iterable: I y IV	Iterador: II y III
D)	Iterable: III y IV	Iterador: I, II
E)	Iterable: IV	Iterador: I, II y III

12. En base al siguiente código, ¿cuál será el valor de la variable `resultado`?

```
1  class A():
2      def funcion_a(self):
3          return 1
4
5  class B(A):
6      def funcion_b(self):
7          valor = self.funcion_a()
8          return 1 + valor
9
10 class C(B):
11     def funcion_a(self):
12         return 2
13
14     def funcion_c(self):
15         valor = self.funcion_b()
16         return 3 + valor
17
18 instancia = C()
19 resultado = instancia.funcion_c()
```

A) 4

B) 5

C) 6

D) 7

E) El código lanza una excepción al calcular el valor de la variable.

13. A partir del siguiente código, indica en qué momento de la ejecución se crea cada elemento contenido en el generador '`mi_generador`':

```
1  def procesar_numero(numero, contexto):
2      print(numero, 'en', contexto)
3      return numero ** 2
4
5  mi_generador = (procesar_numero(numero, 'generador') for numero in range(5))
6
7  for numero in mi_generador:
8      print(numero, 'en for')
```

A) Se crean todos los elementos en la línea 5 cuando se define el generador.

B) Se crean todos los elemento al empezar la iteración del generador en la línea 7.

C) Se crea un elemento en cada iteración del generador mediante el *loop* de la línea 7.

D) Se crea un elemento al ejecutar el `print` de la línea 8.

E) Se crean todos los elementos cuando termina la ejecución del *loop* de la línea 7.

14. Suponiendo que estás implementando una estructura híbrida entre un `Nodo` y una `ListaLigada`, la cual presenta el atributo "`siguiente`" que se encarga de guardar la cadena de nodos que forma la estructura.

    Se te indica que debes implementar el método `__len__` de forma recursiva. Indica el código base y el código recursivo que se deben utilizar para completar este método:

```
def __len__(self):
    # Caso base
    # Caso recursivo
```

Opciones caso base

I.
```
if self.siguiente == None:
    return 1
```

II.
```
if self.siguiente == None:
    return 0
```

Opciones caso recursivo

a.
```
return len(self.siguiente)
```

b.
```
return 1 + len(self.siguiente)
```

A) I y a.

B) I y b.

C) II y a.

D) II y b.

E) No se puede implementar un llamado recursivo, ya que no se llama el método `__len__`.

15. Suponiendo que desaparece el *keyword* `for` de Python y debes recorrer de forma manual los elementos de un iterable, tal como se muestra en el siguiente código.

```
1 for elem in iterable:
2     # código del loop
```

Indica la alternativa que implementa un código equivalente al anterior:

A)
```
while True:
    elem = next(iterable)
    # código del loop
```

B)
```
elem = next(iterable)
while elem:
    elem = next(iterable)
    # código del loop
```

C)
```
while True:
    try:
        elem = next(iterable)
        # código del loop
    except StopIteration:
        break
```

D)
```
while True:
    try:
        elem = next(iterable)
        # código del loop
    except:
        break
```

E) No es posible recorrer un iterable sin el uso del *keyword* `for`.

16. Se desea agregar un elemento a distintas estructuras de datos. Se consideran dos enfoques posibles:

```python
for estructura in [numeros, conjunto]:
    if isinstance(estructura, list):
        estructura.append(7)
        print("Se agregó!")
    else:
        print("No se pudo agregar.")
```

Enfoque 1

```python
for estructura in [numeros, conjunto]:
    try:
        estructura.append(7)
        print("Se agregó!")
    except AttributeError:
        print("No se pudo agregar.")
```

Enfoque 2

Analiza ambos enfoques y selecciona cuáles de las siguientes afirmaciones son **correctas**:

I. En el **Enfoque 1**, se evidencia la filosofía **EAFP** (*Easier to Ask Forgiveness than Permission*).

II. En el **Enfoque 1**, se verifica la validez de la estructura de datos antes de modificarla.

III. En el **Enfoque 2**, se evidencia la filosofía **EAFP** (*Easier to Ask Forgiveness than Permission*).

A) Solo I

B) I y II

C) I y III

D) II y III

E) I, II y III

17. ¿Cuál de las siguientes alternativas es **falsa** respecto a las Excepciones en Python?

A) Un código de Python podría seguir ejecutándose a pesar de levantar múltiples excepciones en este.

B) Todos los tipos de excepciones vistos heredan de la clase `Exception`.

C) El comando `try` se usa para encapsular una sección de código que podría generar errores.

D) El comando `raise` se usa para capturar una excepción.

E) Existen excepciones que se detectan al instante de ejecutar el programa.

18. A partir del siguiente *traceback*, ¿qué provocó el error **KeyError**?

```
Traceback (most recent call last):
  File "menu_diccionario.py", line 11, in <module>
    mostrar_info(p)
  File "menu_diccionario.py", line 7, in mostrar_info
    precio = obtener_valor(plato)
  File "menu_diccionario.py", line 4, in obtener_valor
    return menu[plato]
KeyError: 'Gyoza'
```

A) Se intentó acceder a una llave que no existe en el diccionario.

B) La variable "`menu`" fue definida como lista en lugar de diccionario.

C) La función "`mostrar_info`" presenta un error de sintaxis.

D) Se está intentando retornar un valor **None**.

E) La variable "`plato`" no fue inicializada.

19. ¿Cuál de las siguientes alternativas es **verdadera** respecto a la captura de excepciones?

A) La única forma de capturar múltiples tipos de errores con un solo bloque es usando "**except:**" o "**except Exception:**"

B) No hay forma de capturar excepciones del tipo **SyntaxError** o **IndentationError**, ya que ocurren durante la lectura del programa.

C) No es posible añadir un **finally** a un bloque si no se añadió también un **try**, **except** y **else** en ese bloque.

D) Si un bloque **try** no logra ejecutarse completamente antes de caer en el bloque **except**, todas las acciones ejecutadas dentro del **try** se revertirán automáticamente.

E) Es posible que dentro de bloque **try/except** se levante otra excepción que no pueda ser capturada.

20. En base al siguiente código, ¿cuál es el resultado de su ejecución?

```
traduccion = {"cero": 0, "uno": 1, "dos": 2, "tres": 3}

def funcion(divisor, dividendo):
    try:
        if (not isinstance(divisor, int)) or (not isinstance(dividendo, int)):
            raise TypeError()
        return divisor / dividendo
    except TypeError:
        divisor_2 = traduccion[divisor]
        dividendo_2 = traduccion[dividendo]
        return funcion(divisor_2, dividendo_2)
    except KeyError:
        divisor_3 = traduccion.get(1)
        dividendo_3 = traduccion.get(1)
        return funcion(divisor_3, dividendo_3)
    except ZeroDivisionError:
        return "Incalculable"
    except Exception:
        raise ValueError()

resultado = funcion("cuatro", "cero")
```

A) Un **SyntaxError** no capturado.

B) Un **KeyError** no capturado.

C) Un **ZeroDivisionError** no capturado.

D) Un **ValueError** no capturado.

E) El *string* “Incalculable”.

Preguntas desarrollo

Pregunta 1 - Blocks World (Estructuras de Datos) [6 puntos]

Esta pregunta se divide en dos partes independientes a nivel de código. Ambas te presentan un problema de programación y deberás escribir pseudocódigo para solucionarlo. No es requerimiento que este pseudocódigo siga estrictamente la sintaxis de Python, sino que sea entendible cada paso que siga tu código.

En cada una de estas preguntas debes usar estructuras de datos apropiadas para resolver el problema de forma eficiente, y debes indicar de forma **explícita** las que usas. Las estructuras que puedes usar son: `list`, `tuple`, `namedtuple`, `dict`, `set` y `deque`. Para la parte **1B** se permite usar además `frozenset`, el cual será explicado más adelante.

Enunciado común partes 1A y 1B

Se te ha pedido modelar *Blocks world* usando el contenido de estructuras de datos. *Blocks world* consiste en que se apilan bloques, teniendo una "garra" que puede tomar de a un bloque para luego reubicarlo.

Se usan números para identificar los bloques, estos parten de 1 hasta la cantidad de bloques. Se usará un 0 para modelar cuando no haya un bloque, como es el caso de cuando la garra no tiene un bloque, o para representar el suelo.

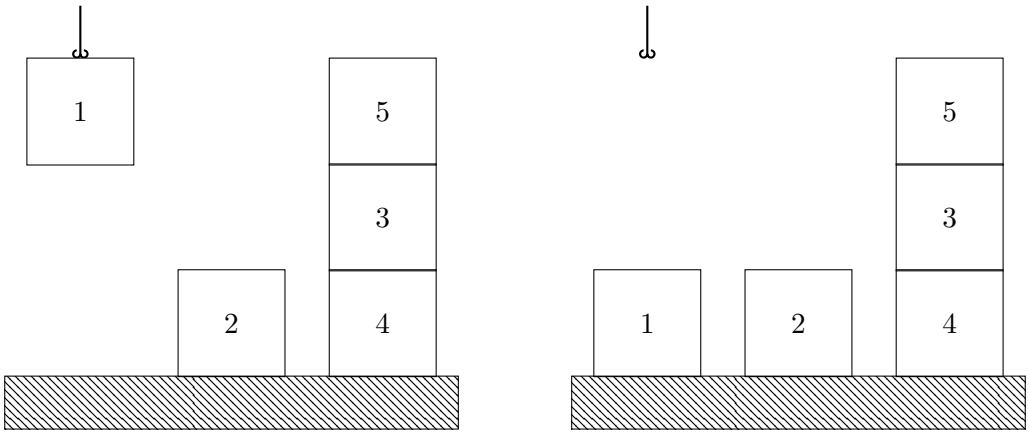

Figura 1: La garra dejando el bloque 1 en el suelo, mientras que los bloques 2 y 4 están sobre el suelo, 3 está sobre 4, y 5 sobre 3

Cada bloque solo puede tener hasta un bloque directamente arriba, pero pueden apilarse formando una torre. Un bloque puede estar ya sea sujeto por la garra, sobre el suelo, o directamente sobre otro bloque. La garra solo puede tomar un bloque que no tenga otro bloque arriba, al tomar un bloque este deja de estar sobre la superficie que estaba y pasa a estar siendo sujeto por la garra. La garra al dejar el bloque solo puede dejarlo en el suelo o sobre un bloque.

Blocks world se modelará usando dos atributos: primero es `garra`, que posee lo que contiene la garra; y segundo es `bloques`, que posee información respecto a los bloques que no están en la garra.

Dado lo anterior, hay dos operaciones: tomar un bloque con la garra, y dejar un bloque que estaba en la garra.

1A: Usando torres

Primero, intentas de modelar *Blocks world* usando torres de bloques. Para el atributo `bloques` de *Blocks world*,usarás el bloque que está en la base para distinguir entre torres, y usando ese bloque puedes acceder a la torre completa que tenga a ese bloque como base.

Deberás programar las siguientes funciones:

- `tomar_bloque(blocks_world, bloque_escogido)`:

  Simula el tomar un bloque con la garra. Recibe `blocks_world` que es la representación de *Blocks world*, y `bloque_escogido` que es el bloque a tomar. Retorna la representación de *Blocks world* modificada dado que la garra tomó el `bloque_escogido`.

- `dejar_bloque(blocks_world, lugar_a_dejar)`:

  Simula el dejar un bloque que actualmente está en la garra. Recibe `blocks_world` que es la representación de *Blocks world*, y `lugar_a_dejar` que es el lugar en el que se dejará el bloque que está en la garra. Retorna la representación de *Blocks world* modificada dado que la garra dejó un bloque en `lugar_a_dejar`.

- `bloques_ordenados(blocks_world)`:

  Revisa si todos los bloques están apilados en una única torre de menor a mayor valor. Recibe `blocks_world`. Retorna un booleano que indica si todos los bloques están apilados en una única torre de menor a mayor valor.

1B: Usando relación de bloque sobre otro

Se te ha mostrado una nueva estructura llamada `frozenset`, la cual funciona igual que un `set`, pero es inmutable, *hasheable* y donde las operaciones de `set` que retornan un `set` ahora retornarán un `frozenset`. Para inicializar un `frozenset`, este recibe como argumento un iterable. Dos `set` se consideran iguales cuando ambos poseen los mismos elementos.

Decides usar esta estructura para guardar el estado en el que se encuentra *Blocks world*. Para esto utilizas la relación de un bloque sobre otro, teniendo en mente que un bloque puede estar sobre otro, o sobre el suelo. Lo anterior lo representarás mediante una estructura que guarde primero el bloque que representa y luego sobre qué está el bloque. Estas relaciones de bloques se guardarán como un `frozenset` en el atributo `bloques` de *Blocks world*.

Por ejemplo, para el *Block world* presentado en la figura 1 (estado final), el `fronzenset` contendría las siguientes relaciones: 1 está sobre 0, 2 está sobre 0, 4 está sobre 0, 3 está sobre 4 y 5 está sobre 3.

Deberás programar las siguientes funciones:

- `tomar_bloque(blocks_world, bloque_escogido)`:

  Simula el tomar un bloque con la garra. Recibe `blocks_world` que es la representación de *Blocks world*, y `bloque_escogido` que es el bloque a tomar. Retorna la representación de *Blocks world* modificada dado que la garra tomó el `bloque_escogido`.

- `dejar_bloque(blocks_world, lugar_a_dejar)`:

  Simula el dejar un bloque que actualmente está en la garra. Recibe `blocks_world` que es la representación de *Blocks world*, y `lugar_a_dejar` el lugar en el que se dejará el bloque que está en la garra. Retorna la representación de *Blocks world* modificada dado que la garra dejó un bloque en `lugar_a_dejar`.

- `bloques_ordenados(blocks_world, blocks_world_esperado)`:

  Revisa si `blocks_world` y `blocks_world_esperado` contienen la misma información.

  Recibe `blocks_world` que es la representación de *Blocks world*, y `blocks_world_esperado`. El segundo contiene la información del *Blocks world* en el mismo formato que está `blocks_world`.

  Retorna un booleano que indica si `blocks_world_esperado` contiene exactamente la misma información que `blocks_world`.

Ambas partes (1A y 1B): Restricciones y consideraciones

- No aparecen ni desaparecen bloques, estos son siempre los mismos. Notar que se sigue considerando que hay un bloque cuando este está en la garra.
- Todos los bloques son cubos y tienen el mismo tamaño.
- El suelo no está restringido, y tampoco hay una forma de indicar la posición de un bloque sobre el suelo (fuera que están a la altura del suelo), es decir, no hay información para saber si un bloque en el suelo está cerca o más a la derecha que otro bloque en el suelo.
- La posición de la garra no es modelada, solo su contenido. Por lo anterior, puedes asumir que la garra no chocará con otros elementos y se ubicará apropiadamente para las operaciones que se te pidan modelar.
- Cuando se te pase el estado de *Blocks world* en una función, puedes asumir que el estado es válido. Por lo anterior, puedes asumir que no habrán dos bloques sobre un mismo bloque.
- Puedes asumir que cuando se realice la acción de tomar un bloque en la garra, se te pedirá un bloque válido. Por lo anterior, puedes asumir que el bloque que se pida tomar no tendrá otro bloque arriba.
- Puedes asumir que para realizar la acción de dejar un bloque con la garra, la garra tendrá un bloque y que el lugar en el que debe dejar el bloque estará disponible.

Pregunta 2 - Modelación OOP [6 puntos]

A partir de la siguiente descripción, debes construir un modelo orientado a objetos que permita capturar los requisitos. El resultado debe mostrarse como un diagrama de clases siguiendo el formato visto en el curso.

> La Deportiva Ciudad Comunitaria (DCC) es un recinto que alberga canchas deportivas para arriendo y uso de la comunidad. Las instalaciones están disponibles para uso de socios de DCC, de deportistas de alto rendimiento, o de clientes regulares que las utilizan de modo recreativo.
>
> Dentro del recinto hay canchas de fútbol, tenis y básquetbol. Cada cancha debe ser arrendada por una persona responsable que debe ser mayor de edad, y posteriormente puede registrar una lista de personas que la utilizará. Cada cancha tiene una hora de inicio del arriendo y una hora de término que debe estar dentro del rango de funcionamiento del recinto DCC.
>
> Cada cancha posee un precio de arriendo, pero el cobro se realiza de manera distinta para cada tipo de cancha. Las canchas de fútbol se pagan por cantidad de personas registradas; las canchas de tenis se arriendan por turnos de uso donde cada turno dura exactamente una hora; mientras que las canchas de básquetbol puede ser solicitadas por fracciones de hora, de acuerdo a su hora de inicio y hora de término.
>
> Los usuarios que son socios de DCC poseen un identificador de socios y reciben un 20 % de descuento cuando son la persona responsable del arriendo de la cancha.
>
> Los deportistas de alto rendimiento pueden usar las canchas gratuitamente si él/ella tiene ese deporte como especialidad, siempre y cuando todas las personas que acompañen al responsable también sean deportistas de alto rendimiento de ese deporte.
>
> Se desea construir un sistema de administración de arriendos para DCC que permita conocer en todo momento los recintos ocupados y disponibles y permita responder preguntas como: cantidad de personas ocupando los recintos, cuántos de ellos son socios, cuántos de alto rendimiento y cuántos son regulares. Puede suponer que los usuarios solo pueden ser: clientes regular, socios, o deportistas de alto rendimiento, pero no más de uno a la vez.

Realice un diagrama de clases que permita modelar esta situación. El diagrama debe indicar al menos: elementos de cada clase como atributos y métodos, relaciones entre clases, y clases y métodos abstractos.

Los nombres de las clases, atributos y métodos deben ser autoexplicativos, de lo contrario deberá agregar una descripción explícita y breve de lo que hace cada uno.

No se solicita escribir código. Puede agregar los supuestos que considere necesarios mientras no contradigan la descripción presentada.