Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-2

Midterm

17 de octubre 2023

Selección múltiple [80 pts]

1. Respuestas

Cada alternativa correcta otorgaba **4 pts**.

1. C
2. B
3. D
4. A
5. Anulada
6. C
7. D
8. A
9. B
10. E
11. D
12. B
13. D
14. A
15. E
16. E
17. B
18. A
19. B
20. C o E

Comentarios

- **Pregunta 5:** Esta pregunta fue anulada porque la afirmación I si es correcta cuando una clase abstracta no posee métodos abstractos en Python. Lo cuál haría que la respuesta D fuera correcta, pero a modo general, la afirmación I es incorrecta, una clase abstracta no se puede instanciar. Así que no existe respuesta correcta en esta pregunta. En consecuencia, todos/as tendrán esta pregunta correcta.
- **Pregunta 20:** En la alternativa E, el mensaje a transmitir era que "Ninguna línea se iba a ejecutar dado ese `raise`", lo cual es incorrecto, la línea 1 y 2 si se ejecutarían. No obstante, decir "No se ejecuta el código completo" si es correcto porque no todas las líneas se ejecutan, solo las primeras 2. En consecuencia, dada dicha ambigüedad en la alternativa E, se consideró como correcta junto con la C, la cual era la correcta inicialmente.

Preguntas desarrollo [40 pts]

Inciso 1 - Modelación OOP (Parte A) [16 pts]

Clase 1 - Deportista

Nombre de la clase: Deportista **(1.0 pts)**.

- Indique si es clase abstracta: Si **(1.0 pts)**.
- Si hereda de una o más clases, indique de cuál o cuáles hereda: ABC (dar 1.0 pts si dice que es abstracta o si dice que herada de ABC).

Atributos. Por cada atributo que añada, indique:

Nombre	Tipo de dato	Property (sí/no)
nombre	`str` **(0.25 pts por nombre + tipo)**	no **(0.25 pts)**
equipo	`str` **(0.25 pts por nombre + tipo)**	no **(0.25 pts)**
edad	`int` **(0.25 pts por nombre + tipo)**	no **(0.25 pts)**
estamina máxima	`int` o `float` **(0.25 pts por nombre + tipo)**	no **(0.25 pts)**
estamina	`int` o `float` **(0.5 pts por nombre + tipo)**	si **(0.5 pts)**

Métodos. Por cada método que añada, indique:

Nombre	Descripción
competir **(0.5 pts)**	Pierde 20 ptos de estamina e imprime el mensaje. **(0.5 pts)**
descansar **(0.5 pts)**	Recupera 50 ptos de estamina. **(0.5 pts)**
entrenar **(0.5 pts)**	Aumenta su estamina máxima. **(0.5 pts)**

Clase 2 - Nadador

Nombre de la clase: Nadador **(1.0 pts)**.

- Indique si es clase abstracta: No **(0.5 pts)**.
- Si hereda de una o más clases, indique de cuál o cuáles hereda: Deportista (0.5 pts)

Atributos. Por cada atributo que añada, indique:

Nombre	Tipo de dato	Property (sí/no)
estilo	`str` **(0.5 pts por nombre + tipo)**	no **(0.5 pts)**

Métodos. Por cada método que añada, indique:

Nombre	Descripción
competir **(0.5 pts)**	Se sobreescribe el método para poder nadar **(0.5 pts)**
entrenar **(0.5 pts)**	Se sobreescribe el método para poder nadar **(0.5 pts)**

Clase 3 - Ciclista

Nombre de la clase: Ciclista (1.0 pts).

- Indique si es clase abstracta: No (0.5 pts).
- Si hereda de una o más clases, indique de cuál o cuáles hereda: Deportista (0.5 pts)

Atributos. Por cada atributo que añada, indique:

Nombre	Tipo de dato	Property (sí/no)
modelo	`str` **(0.5 pts por nombre + tipo)**	Si **(0.5 pts)**

Métodos. Por cada método que añada, indique:

Nombre	Descripción
competir **(0.5 pts)**	Se sobreescribe el método para poder andar en bicicleta **(0.5 pts)**
entrenar **(0.5 pts)**	Se sobreescribe el método para poder andar en bicicleta **(0.5 pts)**

Inciso 1 - Modelación OOP (Parte B) [4 pts]

Para modelar al `Biatleta`, será útil hacer uso de **Multiherencia**, haciendo que la clase herede tanto de `Ciclista` como de `Nadador`. Así, el biatleta tiene tanto los atributos de ambas clases y podrá ejercer ambos deportes. El problema que puede presentarse es al momento de competir, donde puede encontrarse el problema del diamante, ya que ambas superclases a su vez heredan de Deportista. Para solucionarlo, se debe evaluar el uso de `super()` de forma adecuada para evitar llamadas repetidas.

Distribución de puntaje

- **(1.0 pts)** Por mencionar **multiherencia** para modelar al `Biatleta` y que debe heredar de ambas clases `Ciclista` como de `Nadador`..
- **(2.0 pts)** Por mencionar el problema del diamante como posible dificultad.
- **(1.0 pts)** Por mencionar el uso de `super()` para resolver el problema.

Inciso 2 - Análisis de código [20 pts]

Distribución de puntaje

- **(1.0 pts)** Por decir si la afirmación es verdadera o falsa.
- **(4.0 pts)** Por la argumentación que aplica correctamente los términos de *threading* y hace referencia al código. Con el fin de otorgar puntaje parcial a la argumentación, se distribuyó el puntaje del siguiente modo:
  - **(1.0 pts)** Si la argumentación incluye el concepto de *threading* correcto.
  - **(1.0 pts)** Que la argumentación incluya referencia a lo expuesto en el código.
  - **(2.0 pts)** Por una correcta construcción de la argumentación, conectando los conceptos de *threading* y referenciando al código.

Detalle de puntaje por afirmación

1. El atributo "tiempo" debería estar dentro de un *lock*.
   - **(1.0 pts)** Falso.
   - 2 opciones de justificación.

     A) Justificación 1: el tiempo no es un recurso compartido entre los jugadores. Cada uno tiene su propio atributo tiempo que es usado al final para calcular puntaje. Por lo tanto, no se ocupa `lock` para que solo 1 pueda editar el atributo tiempo.
     - **(1.0 pts)** Por hacer mención del concepto de *threading*: recurso o zona crítica, elemento compartido.
     - **(1.0 pts)** Por conectar su argumentación con que el tiempo es un atributo de instancia (no de clase) y/o que no depende de nada global.
     - **(2.0 pts)** Por argumentar que en este caso el tiempo no es un recurso compartido o es un recurso "personalz por eso no necesita el `lock`.

     B) Justificación 2: Cómo se hace `.join()` después del `.start()` para el jugador 1, no hay concurrencia en este código, así que no es necesario controlar el acceso al atributo tiempo mediante un `lock`.
     - **(1.0 pts)** Por hacer mención del concepto de *threading*: `.join()`.
     - **(1.0 pts)** Por conectar su argumentación con que los *threads* no se ejecutan al mismo tiempo.
     - **(2.0 pts)** Por argumentar que como los threads no se ejecutan al mismo tiempo (producto del `.join()`), no afecta el uso o no de `lock`.
2. Las 2 instancias de jugadores no están ejecutando su código concurrentemente.
   - **(1.0 pts)** Verdadero
   - Cómo se hace `.join()` después del `.start()` para el jugador 1, no hay concurrencia en este código, así que no es necesario controlar el acceso al atributo tiempo mediante un `lock`.
     - **(1.0 pts)** Por hacer mención del concepto de *threading*: `.join()`.
     - **(1.0 pts)** Por conectar su argumentación con que los *threads* no se ejecutan al mismo tiempo.

- **(2.0 pts)** Por argumentar que como los threads no se ejecutan al mismo tiempo (producto del `.join()`), no afecta el uso o no de un `lock`.

3. Aunque eliminemos los `.join()` del código, el programa sólo finalizará cuando ambos competidores terminen.
   - **(1.0 pts)** Verdadero si justifica con la opción 1. Falso si justifica con la opción 2.
   - 2 opciones de justificación
     - Opción 1: Los jugadores no son *Daemons*, por lo tanto, el programa principal debe esperar que ellos terminan para finalizar. Por lo tanto, eliminar los `.join()` igual provocará que el programa solo finalice cuando los threads terminen su ejecución.
       - **(1.0 pts)** Por hacer mención del concepto de *threading*: *daemon*.
       - **(1.0 pts)** Por indicar que los jugadores no son *Daemon*.
       - **(2.0 pts)** Por argumentar que el programa sólo finaliza cuando todos los *thread* no *daemon* finalizan y por eso no afecta eliminar los `.join()`
     - Opción 2: Cómo los jugadores no son *daemon*, el programa debe esperar a que ellos finalicen, pero el método `run()` nunca termina. Por lo tanto, el programa nunca finalizará.
       - **(1.0 pts)** Por hacer mención del concepto de *threading*: *daemon*.
       - **(1.0 pts)** Por indicar que los jugadores no son *Daemon* o que hay un ciclo que no finaliza.
       - **(2.0 pts)** Por argumentar que el programa nunca va a finalizar porque los jugadores nunca terminan su ejecución.

4. El código refleja fielmente que los jugadores competirán hasta superar los 500 puntos, luego el primer jugador en notar eso avisará que la competencia terminó y el método `run` finaliza su ejecución.
   - **(1.0 pts)** Falso
   - El método `run` finaliza cuando se levanta una señal (se espera a un `isSet()`), pero dentro del método, jamas se levanta dicha señal cuando se llega a los 500 puntos, solo se hace un `print`. Por lo tanto, el código refleja el desarrollo de la competencia, pero no el final de esta.
     - **(1.0 pts)** Por hacer mención del concepto de *threading*: Eventos y/o levantar señal.
     - **(1.0 pts)** Por indicar que en el código no se hace `set()` de la señal correspondiente y/o que el ciclo solo finaliza cuando se haga `set()` de dicha señal.
     - **(2.0 pts)** Por argumentar que el jugador solo finaliza cuando se levanta una señal, y esta no se levanta nunca. Por eso no refleja fielmente la competencia.