PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IIC2113 - Diseño Detallado de Software (2020-2)

Pauta Exámen

Pregunta 1 [15 puntos totales]

Responda las siguientes afirmaciones declarando si son verdaderas o falsas (1 punto) justificando el por qué de la opción elegida (3 puntos) y citando la fuente de su respuesta (1 punto).

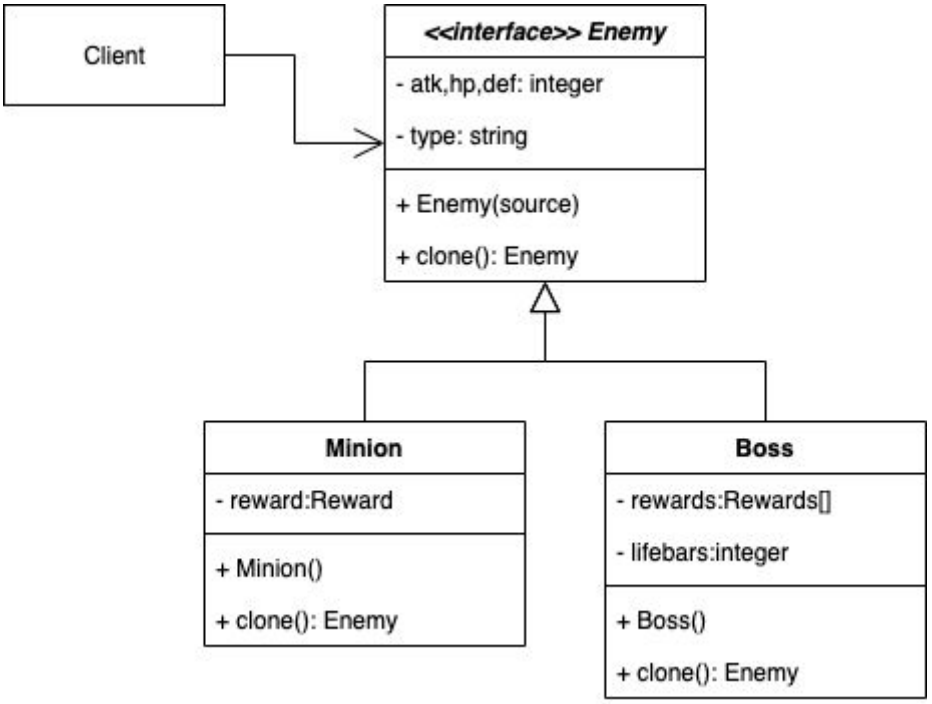
a) No todos los DSL dependen de un lenguaje, los DSL externos son turing completos y por lo tanto no dependen de un lenguaje particular. **[V/F]**
Falso. Existen DSL externos que son turing completos, pero no es intrínseco de todos los DSL externos. La particularidad del lenguaje es que definen su propia sintaxis y podrían ser turing completos o compilados por un sistema externo (como el navegador).
1 punto por responder correctamente que es falso.
2 puntos por explicar que significa DSL externo
1 punto por explicar que algunos son turing completos pero no es requisito para ser DSL externo
1 punto por la fuente: clases

b) La programación reactiva es aquella que usa la librería React.js como base, de tal forma que para hacer programación reactiva en Ruby, hay que importar React.js. **[V/F]**
Falso. La programación reactiva es un paradigma de programación al igual que otros, por lo que expone una forma de programar.
1 punto por responder correctamente que es falso.
2 puntos por explicar que significa paradigma de programación reactiva
1 punto por explicar que React.js es solo una librería que utiliza ese paradigma
1 punto por la fuente: clases

c) En ingeniería inversa, en general a menor nivel de extracción, mayor completitud. **[V/F]**
Verdadero. La completitud, dado un nivel de extracción, se refiere a la cantidad de detalle que se puede obtener. Generalmente es inversamente proporcional al nivel de extracción.
1 punto por responder correctamente que es verdadero
2 puntos por explicar completitud y extracción
1 punto por explicar que son inversamente proporcionales
1 punto por fuente: clases

Pregunta 2 [15 puntos totales]

Se está implementando un juego que define personajes enemigos que serán mostrados en el cliente o escenario del sistema a medida que sean requeridos. Para la definición de los enemigos se ha decidido usar el siguiente diseño:



En base a la información anterior responda las siguientes preguntas:

a) ¿El diseño intenta resolver la creación, la estructura o el comportamiento del modelo? (1 punto) ¿Qué patrón de diseño se está intentando implementar (nombrarlo) y por qué? (3 puntos) ¿Cuál es la ventaja de usar un patrón de diseño? (1 punto). **Responder en un máximo 5 líneas**.
Intenta resolver la creación. Se está usando patrón prototipo ya que en base a un prototipo de enemigo se crean instancias de enemigos clonandolas. La ventaja en este caso partícula es que expone una interfaz común sin la necesidad de acoplar los clones a la clase prototipo. La ventaja del uso de patrones en general es que permiten cumplir de mejor manera con la mantenibilidad del código, en este caso desacoplando dependencias innecesarias entre clases.

b) Si luego de analizar el diseño se descubre que la clase **Minion** no implementa **type** ni **def** de la misma forma que un Boss**,** por lo que al implementarla hay que re-definirlas y/o eliminarlas. ¿Cuál o cuáles Code Smell podrían aparecer? (2 puntos) ¿Por qué? (3 puntos). **Responder en un máximo 5 líneas**.
Podrían aparecer Change Preventers - Divergent Change y Object-Orientation Abusers - Refused Bequest. 1 punto cada uno, puede elegir otro de Object-Orientation Abusers y

otro de Change Preventers, pero debe existir el análisis de los dos casos: no cambiar la clase padre (OO Abusers), o re-definirlas y eliminar (Change Preventers). El primero ya que al usar el patrón prototipo los cambios se harían directamente en la clase Minion, lo que implicaría que habría que re-hacer las dependencias de Boss y la clase padre. El otro camino es no modificar la clase y dejar los atributos que no se usaran, en ese caso el code smell es Refused Bequest pues Minion tiene atributos que no sirven (1.5 puntos cada uno, una buena justificación de otro code smell del mismo tipo se considera correcto, pero debe haber uno de cada uno).

c) Implemente en C# o Ruby las clases Enemy, Minion y Boss representando sus relaciones. Considere también crear una clase Reward vacía para su funcionamiento. El código no debe tener errores de compilación ni de lenguaje. (5 puntos)
Se evalúa caso a caso, pero son 1 punto por cada clase y 2 puntos por las relaciones. Se descuenta por code smell.