TRABAJO PRÁCTICO N°1

Alumno: Oreskovic, Franco Juan

1- El protocolo de la segunda opción es el correcto, ya que este contiene métodos setters para los distintos atributos de la clase Rectangulo y no es necesario manipular directamente los atributos fuera de la definición de la clase.
Además, el constructor de la clase directamente pide argumentos para los 4 atributos, de modo que, una vez instanciada, ya está lista para ser usada. La opción 1, por otro lado, tiene un constructor que no requiere de argumentos, por lo que la instancia no está lista para ser usada ni bien se inicializa -debe tener algún valor inicial por el cómo funciona Wollok, pero ni siquiera sabemos cuáles son-. Se requiere inicializar los atributos de la instancia (y accediendo a estos directamente, lo cual no está tan bueno).

2- La opción correcta es la número 2. En la primera no se le delega la responsabilidad al secretario, sino que es desde la propia clase Jefe que se obtiene el fichero del secretario y se busca la ficha, de modo que pierde bastante sentido el tener un secretario con un fichero si este no puede ser capaz de buscar la ficha por el mismo.
La opción correcta es en la que el secretario sí tiene la capacidad de buscar una cierta ficha en su fichero, de modo que el jefe puede delegarle la responsabilidad, sin tener que preocuparse sobre cómo lleva a cabo esto el empleado. De esta forma, tiene más sentido tener un secretario asociado al jefe.
Según la teoría vista en Programación con Objetos 1, el objeto que tiene la mayor información disponible es el que debe llevar a cabo el comportamiento como tal (en este caso, el secretario y no el jefe).

3- Algo que está mal en las 4 opciones es que en ninguna se hace una validación antes de llevar a cabo la acción que transforma el estado del objeto. Lo que se hace es llevar a cabo dicha acción solo si se cumple una cierta condición, y, sino, no se hace.
Sin embargo, tanto cuando se lleva a cabo como cuando no se lleva a cabo el cambio de estado, en ambos casos el método se ejecuta exitosamente, por lo que el usuario puede creer que la operación se llevó a cabo exitosamente y se dio el cambio de estado cuando en realidad esto no sucedió, pero no existió un error que le advirtiera esto.
En sintesis, el comportamiento solo debería llevarse a cabo exitosamente cuando el objeto tiene la capacidad de llevarlo a cabo, y, cuando no, se debería lanzar una excepción.

Respecto a la opción 1 y 2, estas desaprovechan el polimorfismo al usar en una condición de un if a su propia clase o a un string ligado a su clase.
Los programas en el paradigma de objetos tienen la habilidad de detectar en runtime la clase real de un objeto e invocar a su implementación para el método que se buscaba ejecutar. Por tanto, preguntar sobre la clase de la instancia para llevar a cabo o no una acción es desaprovechar un gran recurso del paradigma.

La opción 3, por su parte, mejora respecto a las otras 2 al dejar que cada subclase concreta tenga su propia implementación del método, de modo que cada una lo lleve a cabo de acuerdo a lo que representan y no se tenga una sola implementación en la que se lleva a cabo el comportamiento de forma A si tenemos una instancia de X o de forma B si tenemos una instancia de Y.
Como existe el polimorfismo, el programa detectará de qué clase es el objeto que quiere ejecutar el método e invocará la implementación correspondiente a esta.

No obstante, también tiene una falencia. En ambas subclases se repite self.setSaldo(self.getSaldo() - unMonto. Nunca está bueno tener código repetido.
Lo que cambia entre ambas es la condición del if del método, por lo que podría mejorarse para que exista un método base en la superclase con el código común entre ambas subclases y que la condición pase a ser un método aparte el cual cada subclase implementa a su manera.

Eso es exactamente lo que se hace en la opción 4, que es la más acertada, ya que acá no hay código repetido. Cada subclase de CuentaBancaria implementa a su manera el método booleano para saber si extraer o no en base a lo que representa la subclase. De esta manera se continua aprovechando el polimorfismo pero también se evita el duplicado de código.

Actividad de lectura #1-

1. ¿Qué significa el acceso directo a las variables? De un ejemplo.

#1-1- Utilizar directamente el atributo de un objeto sin ningún método de por medio (ejemplo: self.atributo u objeto.atributo).
Para esto, se debe acceder al atributo del objeto desde un lugar coincidente con la accesibilidad de dicho atributo (ejemplo: si es privado, solo se puede acceder directamente a este desde la definición de la clase).

2. ¿Qué significa el acceso indirecto a las variables? De un ejemplo.

#1-2- El acceso a las variables se da mediante un método getter para acceder a su valor (ejemplo: objeto.getValor()) y un método setter para modificarlo (ejemplo: objeto.setValor(valor)). Tambien se debe invocar el método desde un lugar coincidente con la accesibilidad de dicho método.

3. Qué ventajas y desventajas presenta cada estrategia referida a los getters y setters.

#1-3-Como aspecto positivo, el método de acceso directo a las variables es mucho más prolijo y comprensible que tener que invocar un método cada vez que se quiere manipular una variable.

Con el método indirecto se pierde la simplicidad y la fácil comprensión antes mencionadas, que no es algo que nos den los getters y setters.
Además, se deben definir estos métodos para las distintas variables, lo cual suma a la interfaz de la clase y la “engorda”.

No obstante, con el método directo, muchos métodos simplemente asumen que el valor de la variable es válido.
Esto causa que, si se quiere introducir, por ejemplo, la inicialización lazy de un atributo, no sea posible debido a que ya se asume que la variable tiene un valor válido.
Según el autor, siempre es recomendable recurrir al método de acceso indirecto y utilizar getters y setters para el acceso a las variables (incluso dentro de la propia definición de la clase e incluso con atributos de accesibilidad privada, usando getters y setters también privados).
Este es conveniente a la hora de usar herencia (ya que, sino, se le debe dar una accesibilidad protected a los atributos para que puedan ser accedidos desde las

subclases. Con acceso indirecto, los atributos tienen accesibilidad privada y accedemos a estos solo mediante getters y setters. Si queremos que un atributo pueda ser accedido y modificado por otras clases, ahí hacemos los getters y setters públicos).

Incluso usando el método de acceso indirecto, el autor no está muy a favor de tener setters públicos para atributos. Prefiere que el valor sea seteado al momento de realizar la instanciación del objeto mediante Creation Parameter Method (pasar el valor que se quiere almacenar en el atributo del objeto al constructor de la clase).

Actividad de lectura #2-

.¿En qué situación es conveniente utilizar el "Creation Parameter Method"?

#2- Consiste en utilizar el constructor de la clase pasándole parámetros a este, los cuales se asignarán a ciertos atributos de la clase (inicialización) (dependiendo de cómo esté armado el constructor). De esta forma, se settean todos los valores que se quiere para una instancia de X clase.
Si usamos siempre el método de acceso indirecto, acá también usaríamos setters dentro del constructor para establecer el estado inicial de la instancia creada.

Si un atributo de un objeto solamente se tiene que inicializar al instanciar a este, pero no se espera que ese atributo pueda ser setteado en otra ocasión que no sea esta etapa de inicialización, para ese atributo no se provee un setter público (y, si no cambia tampoco como parte del comportamiento del objeto, tampoco uno privado).

Actividad de lectura #3-

.¿Cómo se debe proporcionar acceso a variables que referencian a una colección? (Collection Accessor Method)

#3- No es buena idea proporcionar un getter para obtener la colección que tiene almacenada el objeto, ya que esto permite que objetos ajenos puedan realizar alguna modificación sobre la colección sin que el objeto contenedor se entere de esto.
Lo ideal es que la información específica relativa a la colección sea proporcionada por el objeto contenedor mediante métodos de consulta, y que las únicas modificaciones que se puedan hacer sobre las colecciones se hagan como parte del comportamiento del objeto contenedor mediante métodos de orden, en los que siempre es este el que manipula la colección y no otros objetos ajenos.
Estos métodos, al ser sobre colecciones, requieren realizar recorridos sobre las mismas en los que se delegue responsabilidad a los elementos de la colección (estos recorridos pueden realizarse con métodos de colecciones ya existentes, -ej: remove- pero siguen siendo recorridos)
De este modo, las colecciones que almacena un objeto no deberían poder ser modificadas por objetos ajenos ni de forma directa ni indirecta.

Puede ser util definir un método que reciba como parámetro un bloque de código el cual se aplique sobre cada uno de los elementos de la colección (creo que esto es Enumeration Method).

Actividad de lectura #4-

.¿Por qué son necesarios dos métodos para asignar el estado a una propiedad

booleana? (Boolean Property Setting Method)

#4- En vez de tener un setter para el atributo booleano, se usan dos métodos para asignar los dos valores booleanos posibles al atributo. Esto requiere de un método más que simplemente utilizando un setter, pero aporta a la abstracción, prescindiendo de un método muy ligado a la implementación que exponía el cómo está representada la clase para pasar a tener 2 métodos ligados a la lógica de negocios de la clase que no exponen a la clase.

Los métodos pueden llamarse make + estado + () (o toggle, sino).