**1 - Evaluación de protocolos de una clase**

Sea la clase Rectángulo con 4 variables de instancia ( esquinaSuperiorIzquierda ,

esquinaSuperiorDerecha , esquinaInferiorIzquierda y esquinaInferiorDerecha ). Elija uno de los

protocolos presentados a continuación y justifique su elección. Recuerde que el protocolo es el conjunto

de mensajes que entiende una clase o tipo.

Repuesta:

*Ninguna. Porque, por ejemplo,en la opción 2 existe el mensaje "reubicarEsquinaSuperriorIzquierda" es un mensaje ambiguo, será reubicar solo la esquina o toda la figura?. Quizás un mejor nombre seria, reubicarFigura. Si este mensaje no exitiera este seria el que elegiría.*

*Con respecto a la opción 2 tampoco me gusta, los métodos tratan de reubicar cada esquina por su cuenta, cuando en realidad para formar un rectángulo debería estar todas en la misma esquipo.*

**2 – Delegación**

En una oficina hay un jefe que tiene un secretario el cual administra un fichero. Dadas las siguientes

implementaciones seleccione la mejor alternativa y justifique

Respuesta:

*Elegiría la 1, debido a que de esa manera el secretario depende del jefe, y no al revez.*

*De la manera del punto 2 el secretario es independiente del jefe. Quizás si tendría un método llamado serSecretarioDe (Jefe) el planteamiento de la opción 2 seria valida.*

**3 – Polimorfismo**

Existen cuentas bancarias que pueden ser de tipo CajaDeAhorro y CuentaCorriente . Indique los

defectos de cada una de las opciones.

Respuesta:

Opción 1 )

*Lo que hace el método Extrar (UnMonto) no es muy claro, la comparaciones podrían ir en métodos separados para darle mas abtraccion a la comparación.*

*El item selft.Class, lo reemplazaria por un atributo propio de la clase.*

*Y la variable rojo tampoco hacer mucho referencia a lo que es exactamente, es el color de la cuenta, es un estado de la cuenta?, si es así seria un atributo de la clase también y no una variable loca.*

Opción 2 )

*En la opción 2 la comparación también debería extraerse a un método para darle abastraccion. De nuevo la variable rojo, no hace representación a nada, o es un atributo o le cambiaria el nombre.*

Opción 3 )

*En ambas, el menejor de erres por si el if no es cierto*

*De nuevo la comparación del if quizás se podría abtraer mas (por ejemplo: self.puedeExtraer(unMonto)) y ser parte de la superClase*

*Ambas clases sobre-escriben el método extraer, si todas lo hacen entonces quizás no seria bueno que se se haga herecian de ese método.*

Opción 4 )

*De la que mas me gusta, Me hace ruido que ambas sobreescriban el método ChequearSaldoParaExtraccion*

*Se podría ganar un poco de abtraccion en “*self.setSaldo(self.getSaldo() - unMonto)” con un método propio de la case como Self.quitarMonton(unMonton)

**Actividad de lectura #1**

En el libro de Kent Beck Smalltalk Best Practices Patterns se discute la utilización de accessors.

Lea las secciones Direct Variable Access e Indirect Variable Access que se encuentran al final de este

Trabajo práctico.

Luego responda a las siguientes preguntas:

1. ¿Qué significa el acceso directo a las variables? De un ejemplo.
   1. El Acceso Directo a la Variables es la escritura o lectura de una variable directamente sobre la variable, sin pasar por métodos de control de la misma o filtros en el caso. Según el caso, de esa manera por ejemplo.

Class cuadrado {

LadoIzquierdo=1

LadoDerecho=1

LadoSuperior=1

LadoInferior=1

Method cambiarLadoIzquierda {

LadoIquierdo = 3

}

}

1. ¿Qué significa el acceso indirecto a las variables? De un ejemplo.
   1. El acceso Indirecto a variables, es generar funciones o método que su única resposabiliad sea la de cambiar el valor de la variables de la clase. De esta manera se puede tener una idea desde donde proviene la variable dándole una legibilidad mayor al código. Por ejemplo:

Class cuadrado {

private LadoIzquierdo

private LadoDerecho

private LadoSuperior

private LadoInferior

method LadoIzquierdo(nuevaMedida){

self.ladoIzquierdo= NuevaMedida

}

}

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

*La ventajas de usar Getter y Setter, es tener el control total del estado de nuestra clase. Evitando que la clase quede en estado erróneo.*

*La desventaja es que por cada atributo existente en nuestra clase, se tiene que configurar un setter y un getter, o sea que se duplicaría la cantidad de método por cada atributo existente.*

Class cuadrado {

private LadoIzquierdo

private LadoDerecho

private LadoSuperior

private LadoInferior

method LadoIzquierdo(nuevaMedida){

self.ladoIzquierdo= NuevaMedida

}

method LadoIzquierdo(){

return self.LadoIzquierdo

}

**Actividad de lectura #2**

En base al siguiente fragmento, responda: ¿en qué situación es conveniente utilizar el "Creation

Parameter Method"?

Respuesta:

Es conveniente usarlo cuando en la instancia de generar los Getter no es muy compleja. Ya que si este método debería realizar la asignación de los parámetro y a la vez validar cada uno de ellos.

**Actividad de lectura #3**

En base al siguiente fragmento, responda: ¿cómo se debe proporcionar acceso a variables que

referencian a una colección?

*Respuesta:*

*Siempre el acceso a las colecciones debe usando Acceso a Variables indicectamente (AVI).*

*Haciendo que lo getter privador para que no se tenga no tengo acceso a las colecciones directamente y estas sean manejadas por externos.*

*De esta manera se debe hacer un método de acceso “personalizado “ para cada colección que se tenga dentro de la clase.*

*Entonces se pueden concatener claves para que las colecciones sea manejadas por otros. Por ejemple*

*Escalera.quitarEscalones, o Escalera.AgregarEscalones,*

**Actividad de lectura #4**

En base al siguiente fragmento, responda: ¿por qué son necesarios dos métodos para asignar el

estado a una propiedad booleana?

*Respuestas:*

*Para dividir las responsabilidades justamente para darle expresividad al cambio de estado de una booleno. Además el nombre también es importante, debido a que hay que dejar en claro que no es una pregunta, o un seteo. Una solución es anteponer el Make adelentar del método para la afirmación y la negación lo contrario por ejemeplo MakeVisible, or MakeInvisible.*