Descripción de reglas de diseño

Es esta sección se describen algunas reglas o heurísticas de diseño que nos permiten crear buenos modelos. Hacer buenos diseños nos permite que nuestros programas sean mantenibles, entendibles y estén preparados para el cambio.

Mapeo con dominio de problema

En esta sección se describen algunas reglas o heurísticas de diseño a tener en cuenta cuando diseñamos con objetos para mantener un mapeo con el dominio de problema.

1. Cada ente del dominio de problema debe estar representado por un objeto.
    a. Las ideas son representadas con usa sola clase.
    b. Los entes pueden tener una o más representaciones en objetos, depende de la implementación.
    c. La esencia del ente es modelado por los mensajes que el objeto sabe responder.
2. Los objetos deben ser cohesivos representando responsabilidades de un solo dominio de problema. Mientra más cohesivo es un objeto más reutilizable es.
3. Se deben utilizar buenos nombres, que sinteticen correctamente el conocimiento contenido por aquello que están nombrando.
    a. Los nombres son el resultado de sintetizar el conocimiento que se tiene de aquello que se está nombrando.
    b. Los nombres que se usan crean el vocabulario que se utiliza en el lenguaje del modelo que se está creando.

4. Las clases deben representar conceptos del dominio de problema.
    a. Las clases no son módulos ni componentes de reuso de código.
    b. Crear una clase por cada "componente" de conocimiento o información del dominio de problema.
    c. La ausencia de clases implica ausencia de conocimiento y por lo tanto la imposibilidad del sistema de referirse a dicho conocimiento.

Subclasificación

En esta sección se describen algunas reglas o heurísticas de diseño para utilizar bien la subclasificación.

1. Se deben utilizar clases abstractas para representar conceptos abstractos.
    a. Nunca denominar a las clases abstractas con la palabra Abstract. Ningún concepto se llama "Abstract…"
2. Las clases no-hojas del árbol de subclasificación deben ser clases abstractas.
3. Evitar definir variables de instancia en las clases abstractas porque esto impone una implementación en todas las subclases.
4. El motivo de subclasificación debe pertenecer al dominio de problema que se está modelando.
5. No se deben mezclar motivos de subclasificación al subclasificar una clase.

Polimorfismo, código repetido y creación de objetos

En esta sección se describen algunas reglas o heurísticas de diseño sobre el polimorfismo, el código repetido y la creación de objetos.

1. Reemplazar el uso de if con polimorfismo.
    a. El if en el paradigma de objetos es implementado usando polimorfismo.
    b. Cada if es un indicio de la falta de un objeto y del uso del polimorfismo.
2. Código repetido refleja la falta de algún objeto que represente el motivo de dicho código.
    a. Código repetido no significa "texto repetido" sino que significa patrones de colaboraciones repetidas.

    b. Reificar ese código repetido y darle un significado por medio de un nombre.
3. Un objeto debe estar completo desde el momento de su creación.
    a. El no hacerlo abre la puerta a errores por estar incompleto, habrá mensajes que no sabe responder.
    b. Si un objeto está completo desde su creación, siempre responderá los mensajes que definió.
4. Un objeto debe ser válido desde el momento de su creación.
    a. Un objeto debe representar correctamente el ente desde su inicio.
    b. Junto a la regla anterior mantienen el modelo consistente constantemente.

Evitar usar None, Objetos inmutables, modelar la arquitectura del sistema

En esta sección se describen algunas reglas o heurísticas de diseño sobre la necesidad de evitar el uso del objeto None, utilizar objetos inmutables y modelar la arquitectura del sistema.

1. No utilizar None
    a. None no es polimórfico con ningún objeto.
    b. Por no ser polimórfico implica la necesidad de poner un if lo que abre la puerta a errores.
    c. None es un objeto con muchos significados por lo tanto poco cohesivo.
    d. Las dos reglas anteriores permiten evitar usar None.
2. Favorecer el uso de objetos inmutables.
    a. Un objeto debe ser inmutable si el ente que representa es inmutable.
    b. La mayoría de los entes son inmutables.
    c. Todo modelo mutable puede ser representado por un inmutable donde se modele los cambios de los objetos por medio de eventos temporales.
3. Evitar el uso de setters (un setter es un mensaje que cambia el valor de una variable)
    a. Para aquellos objetos mutables, evitar el uso de setters porque estos pueden generar objetos inválidos.

b. Utilizar un único mensaje de modificación.

4. Modelar la arquitectura del sistema.
    a. Crear un modelo de la arquitectura del sistema (subsistemas, etc.)
    b. Otorgar a los subsistemas la responsabilidad de mantener la validez de todo el sistema (la relación entre los objetos).
    c. Otorgar la responsabilidad a los subsistemas de modificar un objeto por su impacto en el conjunto.