

Objetos I



Conceptos:

- Tipo: definición de un dato, en Java puede ser interface o clase
- Modelo: abstracción de los elementos de un sistema, indica qué haces no cómo lo haces
- Clase: es una descripción abstracta de un conjunto de objetos que comparten los mismos atributos (estado interno) operaciones (mensajes a los que responden los objetos) relaciones y semántica
 - agrupan comportamiento común
 - definen la forma de sus instancias
 - crean objetos que son instancias de ellas
 - todas las instancias de una clase se comportan de la misma manera
 - cada instancia mantiene su propio estado
- asociación: una clase conoce a otra
- generalización: una clase hereda de otra
- Objeto: abstracción de una entidad del dominio del problema, un objeto posee...
 - identidad: para distinguir uno de otro
 - conocimiento: en base a sus relaciones con otros objetos y su estado
 - Comportamiento: conjunto de mensajes que un objeto sabe responder
- Mensaje: tiene nombre y parámetros. para poder enviar un mensaje a un objeto hay que conocerlo, al enviarle un mensaje a un objeto este responde activando el método asociado a ese mensaje
- Método: es la contraparte funcional del mensaje. expresa la forma de llevar a cabo la semántica propia de un mensaje particular. un método puede modificar el estado de otro objeto y enviar mensajes a otros objetos
- Conocimiento: para que un objeto puede conocer a otro lo debe poder nombrar, se hace un "binding"

- conocimiento interno: variables de instancia
- conocimiento externo, parametros
- conocimiento temporal: variables temporales

Encapsulamiento:

- Encapsulamiento: es la cualidad de los objetos de ocultar los detalles de implementación y su estado interno del mundo exterior,
 - esconde detalles de su implementación
 - protege el estado interno de los objetos
 - un objeto solo muestra su cara visible por medio de su protocolo
 - los métodos y su estado quedan escondidos para cualquier otro objeto, es el objeto que decide que se publica
 - facilita modularización y reutilización

Herencia

¿para qué usar jerarquiar?

entender mejor el dominio

reducir el código

aprovechar el polimorfismo

Asociaciones entre clases

Asociación: relacion de conocimiento

Generalización: relaciones de herencia

Interfaz: relación entre una clase interfaz y otra que la implementa

propiedades

rol: caras que presentan las clases

navegabilidad: limita la navegación

visibilidad

a. publica: (+)

b. protegida: (#)

c. privada: (-)

agregación

asociación especial todo/parte una o más clases son partes del todo

composición: fuerte sentido de posesión y tiempo de vida coincidentes de las partes en el conjunto

Generalización:

representa la herencia en objetos se puede tener herencia simple y múltiple.



SOLID

- Single Responsibility Principle: una clase debería tener una única responsabilidad
- Open-Closed Principle: las entidades deberían estar abiertas a la extensión pero cerradas a la modificación. una nueva funcionalidad no debe modificar el diseño existente
- Liskov substitution principle: functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it
los objetos de un programa deben ser intercambiables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa.
correcto uso de herencia y polimorfismo
- Interface segregation principle: las clases que tienen interfaces voluminosas, son clases cuyas interfaces no son coesivas. las clases no deberían implementar métodos que no usen.
si las interfaces tienen mucho protocolo, las clases que lo implementen se ven obligadas a implementar todos los métodos
- dependency inversion principle:
depend upon abstractions not concretions
los módulos de alto nivel de abstracción no deben depender de los de bajo nivel
las abstracciones no deben depender de los detalles, los detalles deben depender de las abstracciones

Heurísticas

Experto en información

asigna una responsabilidad al experto en información expresa la intuición de que los objetos tienen cosas relacionadas con la información que tienen.

Creador

asignar a la clase B la responsabilidad de crear una instancia de A

B contiene objetos A

B registra instancias de A

B tiene los datos para inicializar objetos A

B usa el objeto A de forma exclusiva

la intención de creador es encontrar una clase que necesite conectarse al objeto creador

Controlador

asigna la responsabilidad de manejar eventos del sistema a una clase que representa

sistema global

x ej: quien maneja compras y clientes? el objeto librería

la intención del controlador es encontrar manejadores de los eventos del sistema
sin recargar de responsabilidad a un solo objeto y manteniendo la alta cohesión

Bajo Acoplamiento

pocas dependencias entre clases

una clase que se asocie con muchas otras tiene muchas asociaciones baja la reutilización

el alto acoplamiento dificulta el entendimiento y complica la propagación de cambios en el diseño

no se puede considerar como criterio aislado

Alta cohesión

asigna responsabilidad de manera que la cohesión(fuerza con la que se relacionan las responsabilidades de un objeto) sea importante/fuerte

la cohesión es una medida de la fuerza con la que se relacionan las responsabilidades de un objeto y la cantidad de ellas

la idea es que cada clase tenga responsabilidades bien definidas no por tener pocos métodos sino por tener áreas

ventajas: clases más fáciles de mantener, entender y reutilizar

el nivel de cohesión no se puede considerar de manera aislada a otras responsabilidades y otras heurísticas como experto y bajo acoplamiento

Preguntas multiple choice Teoría:

Cuando un objeto **a** de la clase **A** recibe un mensaje **m**

Seleccione una:

- a. Le pide a la clase que ejecute ese código en la misma clase. La clase devuelve el resultado
- b. Busca en **A** el método correspondiente para responder a **m** y lo ejecuta ✓
- c. Ejecuta un procedimiento interno que determina que hacer con el mensaje

Cuando un objeto recibe un mensaje

Seleccione una:

- a. Ejecuta el método que corresponde al mensaje recibido ✓
- b. Decide que variables puede modificar para cumplir con la acción que le requieren
- c. Invoca un procedimiento que decide cual es la acción a realizarse

A los objetos de nuestra aplicación los crean...

Seleccione una:

- a. Los objetos que pertenecen a una clase especial que controla el flujo del código mmmm raro
- b. Un conjunto de procedimientos especiales que se ejecutan al comenzar la aplicación
nunca se habla de ningún procedimiento especial ni objeto especial
- c. Otros objetos de acuerdo a la lógica de la aplicación ✓

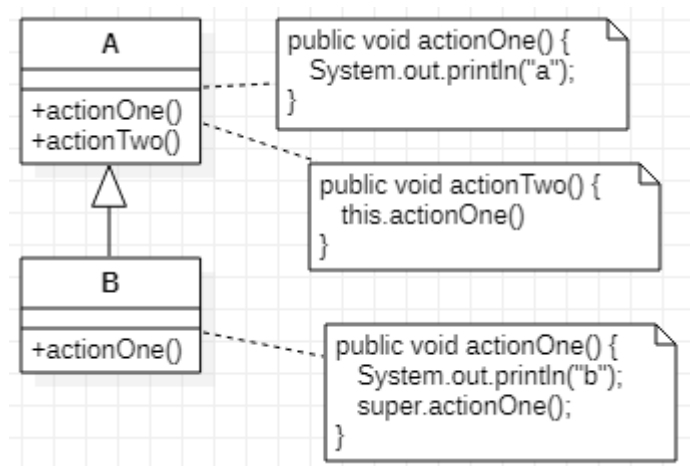
Los objetos tienen variables de instancia que permiten caracterizar su estructura.

Seleccione una:

- a. Los procedimientos que escribimos pueden estar autorizados a acceder a datos de otros procedimientos
- b. Las variables de instancia están encapsuladas y solo son accesibles por los métodos que ejecuta el objeto. Para acceder una variable del objeto A, necesitamos enviarle un mensaje. ✓
- c. Los objetos acceden a las variables de otros objetos en una forma semejante a los registros.

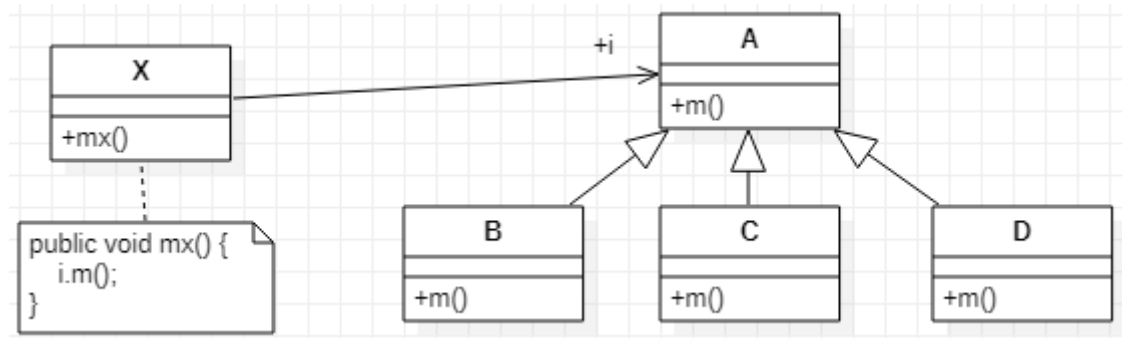
eso si sus atributos son públicos, si se puede acceder a través de los mensajes que podemos mandarle al objeto en particular

Sabiendo que la expresión `System.out.println("algo")` imprime un string por la salida estándar, y dado el siguiente diseño, seleccione la afirmación correcta de la lista:



Seleccione una:

- a. La expresión `(new B()).actionOne()` imprime "b" y luego "a" por la salida estándar. ✓
el mensaje `actionOne()` enviado a B ejecuta el código de abajo y primero imprime "b", luego el `super.actionOne()` indica que tiene que enviar ese mensaje a la clase padre de B entonces ejecuta el código de arriba que imprime "a"
- b. La expresión `(new B()).actionOne()` imprime "a" por la salida estándar.
- c. La expresión `(new B()).actionOne()` imprime "a" y luego "b" por la salida estándar.
- d. La expresión `(new B()).actionOne()` imprime "b" por la salida estándar.



Supongamos que tenemos una clase **A**, con sub-clases **B**, **C** y **D**. En todas ellas tenemos una implementación del método **m()**.

Supongamos que tenemos también la clase **X** con una variable de instancia **i** de tipo **A**.

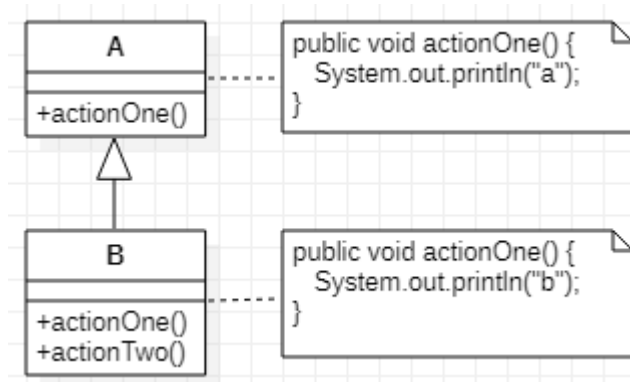
Supongamos que como respuesta a un mensaje enviado a una instancia de **X** se ejecuta el método de **mx()**

Cuando se ejecuta la expresión **i.m()** en dicho método:

Seleccione una:

- a. Implicará que se ejecute el método **m()** de la clase **A**
no necesariamente, podría ser de cualquiera de las clases B C o D y sería otro método distinto al de A
- b. El objeto receptor **x** decide que método debe ejecutarse chequeando los tipos correspondientes
el objeto x no debe encargarse de eso
- c. Implicará que se ejecute el método **m()** correspondiente a la clase a la que pertenece el objeto actualmente apuntado por la variable **i** ✓
dependiendo de si es B C o D se ejecutara el código correspondiente

Sabiendo que la expresión `System.out.println("algo")` imprime un string por la salida estándar, y dado el siguiente diseño, seleccione la afirmación correcta de la lista:



Seleccione una:

- a. La expresión (new B()).actionOne() imprime "b" por la salida estándar. ✓
- b. La expresión (new B()).actionOne() imprime "a" por la salida estándar.
- c. La expresión (new B()).actionOne() imprime "a" y luego "b" por la salida estándar.
- d. La expresión (new B()).actionOne() imprime "b" y luego "a" por la salida estándar.

Decimos que en un lenguaje de programación orientado a objetos existe polimorfismo cuando


Seleccione una:

- a. un objeto **o** puede mandar mensajes diferentes a otros objetos
 - b. puedo elegir diferentes implementaciones del mismo método **m()** según me interese
 - c. el mensaje **m()** puede ser recibido por objetos de clases diferentes. ✓
- esto, el polimorfismo puede darse en objetos que comparten la misma clase padre o que implementan la misma interface.


Seleccione la afirmación correcta

Seleccione una:

- a. Un tipo en un lenguaje orientado a objetos es un conjunto de firmas de métodos ✓
- b. Las interfaces en Java son el mecanismo para dar tipo a las clases ✗
las interfaces pueden estar definiendo un tipo
- c. En un lenguaje de programación orientado a objetos las clases no constituyen tipos, solo las interfaces lo hacen ✗
las clases y las interfaces constituyen tipos

d. Java es un lenguaje orientado a objetos sin tipos porque solo tiene clases 


kjjjj

e. En un lenguaje de programación orientado a objetos, un tipo es lo mismo que una clase 

un tipo esta definido por una clase, no es lo mismo


Seleccione la afirmación correcta

Seleccione una:

- a. El mensaje iterator() transforma la colección que lo recibe en un objeto Iterator que se puede recorrer.
- b. El mensaje iterator() que entienden las colecciones retorna un objeto que abstrae a forma en la que se recorre la colección. 
- c. El mensaje iterator() que entienden las colecciones, recibe como parámetro una expresión lambda y la ejecuta con todos sus elementos.
- d. El mensaje iterator() recorre la colección que lo recibe.

Seleccione la afirmación correcta

Seleccione una:

- a. Es importante testear temprano, y tanto como sea el riesgo del artefacto a testear 

es importante hacer un Test Driven Development para evitar fallas a futuro, y si es un artefacto de riesgo por ejemplo algun instrumento médico habria que evitar las fallas aún más

- b. Es importante testear ni bien se termina de escribir el programa, y con tanta cobertura como sea posible.
- c. Es importante testear todos los métodos hasta alcanzar una cobertura mayor al 80% del código.
- d. Es importante diseñar bien el programa y chequearlo entre pares para evitar escribir tests de unidad.

Seleccione la afirmación correcta



Seleccione una:

- a. En java, las colecciones solo admiten como contenido instancias de clases que implementan interfaces.
no solo interfaces
- b. En Java, es recomendable que todos los objetos en una colección sean de la misma clase.
- c. En Java, es recomendable que todos los objetos en una colección compartan un tipo. ✓
es recomendable
- d. En Java, solo es posible agregar a una colección instancias de una misma clase, o de clases que comparten alguna superclase.



Seleccione una:

- a. Para escribir tests de particiones equivalentes identifico particiones, y elijo valores representativos dentro y fuera de cada partición para usarlos en los tests. ✓
- b. Para escribir tests de particiones equivalentes debo asegurarme de incluir en todas las particiones al valor null, al mínimo, y al máximo.
- c. Para escribir tests de particiones equivalentes, identifico todos los valores posibles y los separo en particiones de tamaño equivalente para usarlas en mis tests.
- d. Para escribir tests de particiones equivalentes identifico particiones, y elijo tantos valores de test de cada partición como complejidad tenga la misma (más compleja, más valores).



Seleccione la afirmación correcta

Seleccione una:

- a. Para filtrar una colección en Java, es recomendable utilizar el protocolo de streams. ✓
- b. Para filtrar una colección en Java, es recomendable utilizar un iterador y una colección adicional en la que se acumula el resultado.
- c. Para filtrar una colección en Java, le envío en mensaje stream() con una expresión lambda como parámetro.
no es así la sintaxis
- d. Para filtrar una colección en Java, utilizo la librería de Pipes and Filters que ofrecen los iteradores. ✗

Las Heurísticas para Asignación de Responsabilidades (HAR) son útiles para:

Seleccione una:

- a. Decidir que clases candidatas forman el Modelo Conceptual o del Dominio
- b. Mejorar el Diagrama de Clases del Diseño
- c. Decidir que objeto dentro del Sistema es responsable o receptor de una operación ✓

En Reuso de código (Herencia):

Seleccione una:

- a. Extendemos pensando que la nueva subclase puede redefinir y anular lo que hereda y no corresponde a su funcionalidad
- b. Extendemos pensando en "Herencia de Estructura" para cumplir con "Is-a"
- c. Extendemos pensando en "Herencia de comportamiento", para cumplir con "Is-a" ✓
- d. Extendemos pensando que usamos sólo lo que nos sirva, que no heredamos atributos y comportamiento de la cadena de superclases

En UML, la relación de conocimiento entre objetos o instancias de clases:

Seleccione una:

- a. se modela con una asociación hacia el/los objetos que se conocen, agregando en el final de la asociación nombre (rol) y multiplicidad. ✓
- b. debe ser siempre bidireccional, es decir, navegable hacia las dos clases

c. se modela con una asociación hacia el/los objetos que se conocen, agregando en el medio de la asociación un nombre y multiplicidad.

d. se representa como un atributo que identifica al/los objetos que se conocen

En los Contratos de Operaciones, las poscondiciones:

Seleccione una:

a. describen el estado del sistema antes de ejecutarse la operación, utilizando conceptos del modelo conceptual o del Dominio

b. describen el estado y cambios del sistema después de ejecutarse la operación, utilizando conceptos del modelo conceptual o del Dominio ✓

c. muestran cómo se ejecutan las acciones dentro de la operación

d. muestran el estado del Sistema durante la ejecución de la operación

En Reuso de código (Herencia Vs. Composición):

Seleccione una:

a. El uso de composición entre objetos genera más alto acoplamiento que el uso de herencia

b. El uso de herencia entre objetos genera más bajo acoplamiento que el uso de composición

c. Cuando la estructura de la nueva subclase es como la de la superclase, se aconseja usar herencia, no composición. El comportamiento heredado que no sirva, puede anularse o redefinirse.

d. El reuso por composición, permite usar al objeto a través de su protocolo, sin necesidad de tener que conocer su implementación ✓

Seleccione la afirmación correcta

Seleccione una:

a. ECMAScript es un lenguaje basado en clases si se lo usa en el servidor, y basado en prototipos si se lo usa en el navegador.

b. ECMAScript es un lenguaje basado en clases si se lo usa en el navegador, y basado en prototipos si se lo usa en el servidor.

c. Al igual que Java y Smalltalk, que son lenguajes orientados a objetos basados en clases, ECMAScript es también basado en clases.

d. A diferencia de Java y Smalltalk, que son lenguajes orientados a objetos basados en clases, ECMAScript es basado en prototipos. ✓

Seleccione la afirmación correcta

Seleccione una:

- a. En Smalltalk todo se implementa con objetos y está abierto a modificación. Incluso lo que comúnmente conocemos como estructuras de control (como el if, while, etc.) se implementa como envíos de mensajes a objetos. ✓
- b. Smalltalk hace una clara diferencia entre las partes de nuestra aplicación implementadas en objetos, por nosotros, y los elementos base del lenguaje (por ejemplo, librerías) que han sido implementados combinando paradigmas (objetos, funcional, procedural).
- c. En Smalltalk todo se implementa con objetos. Sin embargo, la implementación de los objetos de las librerías base (como Boleanos, Strings, colecciones, etc.) no puede modificarse.
- d. En Smalltalk casi todo se implementa con objetos. Los tipos primitivos (Integer, String, etc.) y las estructuras de control no.

Seleccione la afirmación correcta

Seleccione una:

- a. En ECMAScript, cada objeto hereda comportamiento de su prototipo. El estado no se hereda.
- b. En ECMAScript, cada objeto hereda comportamiento y estado de su prototipo. ✓
- c. En ECMAScript, cada objeto hereda estado de su prototipo. El comportamiento no se hereda.
- d. Al no ser basado en clases sino en prototipos, ECMAScript no implementa el concepto de herencia.

Seleccione la afirmación correcta

Seleccione una:

- a. ECMAScript puede utilizarse tanto como un lenguaje dinámico, en el que no es necesario indicar explícitamente el tipo de las variables o como un lenguaje fuertemente tipado.
- b. ECMAScript es un lenguaje dinámico, en el que no se indica explícitamente el tipo de las variables. ✓
- c. ECMAScript es un lenguaje fuertemente tipado, en el que se declara el tipo de cada variable y parámetro, utilizando clases.

d. ECMAScript es un lenguaje fuertemente tipado, en el que se declara el tipo de cada variable y parámetro, utilizando interfaces.

Seleccione la afirmación correcta:

Seleccione una:

a. Smalltalk es un lenguaje fuertemente tipado, en el que se declara el tipo de cada variable y parámetro, utilizando clases.

b. Smalltalk puede utilizarse tanto como un lenguaje dinámico, en el que no es necesario indicar explícitamente el tipo de las variables o como un lenguaje fuertemente tipado.

c. Smalltalk es un lenguaje fuertemente tipado, en el que se declara el tipo de cada variable y parámetro, utilizando interfaces.

d. Smalltalk es un lenguaje dinámico, en el que no se indica explícitamente el tipo de las variables. ✓