POO metodología de desarrollo de aplicaciones en la cual las mismas se organizan como colecciones de cooperativas de objetos, las cuales representan una instancia de alguna clase.

ABSTRACCION aísla un elemento de su entorno. Se enfoca en la visión externa de un objeto, separa el comportamiento especifico, quita las propiedades y acciones de un objeto para dejar solo aquellas necesarias. Es clave para diseñar un buen software.

¿Qué características podemos abstraer de los animales?

ENCAPSULAMIENTO ocultar datos de un objeto de modo que solo se pueda cambiar mediante las operaciones definidas para ese objeto.

Empaquetamiento -> objetos aislados desde el exterior.

HERENCIA creación de una clase a partir de otra existente, obteniendo características y métodos lo que nos permite crear objetos derivados a partir de objetos bases = reutilizar código.

El concepto de herencia conduce a una estructura jerárquica de clases o estructura de árbol donde cada clase tiene ->

->solo una clase padre llamada SUPERCLASE: puede tener cualquier número de SUBCLASES.

->la clase hija se llama SUBCLASE: puede tener solo una superclase.

Ventajas: evita la duplicidad y favorece la reutilización de código, las subclases utilizan el código de la superclase. Facilita el mantenimiento y extensión de las aplicaciones que diseñamos.

Herencia y ABSTRACCION se puede pensar en una jerarquía de clases como la definición de conceptos abstractos en lo alto de la jerarquía ->

->Las subclases no están limitadas al estado y comportamiento provisto por la superclase si no que pueden agregar variables y métodos además de los que ya heredan.

->Las clases hijas también pueden sobrescribir los métodos que heredan.

POLIMORFISMO habilidad de un identificador para poseer varios significados diferentes: varias formas de responder el mismo mensaje. Ligado a la herencia.

Formas ->

->Sobre-carga de métodos: los mensajes se diferencian en los parámetros.

->Sobre-escritura de métodos: un hijo sobrescribe un método de la clase padre.

->Vinculación dinámica: herencia.

HERENCIA Y POLIMORFISMO

->Palabra reservada SUPER se utiliza para invocar métodos de la superclase. En el caso de los constructores debe ser la primera línea en los mismos.

El modificador de acceso PROTECTED en una clase indica que la variable instanciada puede ser leída desde las subclases.

El compilador verifica el tipo de la referencia, no el tipo del objeto.

Los métodos sobrescritos de las subclases tienen precedencia sobre los métodos de las subclases. La búsqueda del método comienza al final de la jerarquía, entonces la última redefinición de un método es la que se ejecuta primero.

OBJETO entidad autónoma que contiene atributos y comportamiento.

Se combinan datos y la lógica de programación. Tienen estado: sustantivos, y comportamiento: verbos.

CLASE plantilla para la creación de objetos.

Cada clase es un modelo que define un conjunto de variables (atributos) y métodos (comportamiento).

INSTANCIA cada objeto creado a partir de una clase.

MODIFICADOS DE ACCESO

->PUBLIC ofrece la máxima visibilidad. Una variable, método o clase será visible desde cualquier clase.

->PRIVATE restringe el uso de método o atributo al interior de la misma clase.

->PROTECTED visibles para las clases del mismo paquete y subclases.

->DEFAULT visibilidad para clases del mismo paquete.

VARIABLES DE INSTANCIA non-static. Son los atributos que tienen los objetos -> son únicos para cada instancia.

VARIABLES DE CLASE se declaran con el modificador static para indicarle al compilador que hay exactamente una copia de la variable, y es compartida por todas las instancias. Pertenecen a la clase y no a los objetos.

VARIABLES LOCALES la determinación viene desde la ubicación en donde la variable fue creada, es decir, local al método. Solo es visible al método donde fue declarada y no puede ser accedida desde el resto de las clases.

JAVA VIRTUAL MACHINE –JVM-

->SECCIONES

->Zona de datos: donde se almacenan las instrucciones del programa, las clases con sus métodos y constantes. No se puede modificar en tiempo de ejecución.

->Stack: el tamaño se define en tiempo de compilación y es estático en tiempo de ejecución. Aquí se almacenan las instancias de los objetos y los datos primitivos.

->Heap: zona de memoria dinámica. Almacena los objetos que se crean.

->TAREAS PRINCIPALES

->Reservar espacio en memoria para los objetos creados.

->Liberar la memoria no usada. (garbage collector)

->Asignar variables a registros y pilas.

->Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos.

->Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java. Por ejemplo: no se permiten realizar ciertas conversiones (casting) entre distintos tipos de datos, las referencias a arrays son verificadas en el momento de la ejecución del programa.

GARBAGE COLLECTOR ->

->Es un proceso de baja prioridad que se ejecuta dentro de la JVM.

->Técnica por la cual el ambiente de objetos se encarga de destruir y asignar automáticamente la memoria heap.

->El programador no debe preocuparse por la asignación y liberación de memoria, ya que el entorno la asignara al crear un nuevo objeto y la librera cuando nadie la esté usando.

->Un objeto podrá ser "eliminado" cuando desde el slack ninguna variable haga referencia al mismo.

TIPO OBJETO son instancias de clases, que pueden estar predefinidas o definidas por el programador. Cuando un objeto es instanciado utilizando el operador new, se retorna una dirección de memoria que contiene una referencia a una variable.

CLASES "WRAPPER" (envoltorio) clases diseñadas para ser un complemento de los tipos primitivos siendo los únicos elementos de Java que no son objetos. Los tipos primitivos siempre se pasan como argumento a los métodos por valor, mientras que los objetos se pasan por referencia. No hay forma de modificar en un método un argumento de tipo primitivo y que esa modificación se transmita al entorno que hizo la llamada a menos que se utilice un Wrapper, objeto cuya variable miembro es el tipo primitivo que se quiere modificar.

->Boxing: realizar una conversión automática del tipo primitivo a su equivalente en objeto. Método estático valueOf()

FOR EACH es una estructura especializada en recorrer los elementos que contiene una variable, es por eso que está especialmente indicado para ver o recorrer todos los elementos de un array. Esta variable cambia con cada repetición, de manera que va mostrando los distintos elementos del array.

Un método declarado con la palabra reservada STATIC nos indica que se puede invocarlo sin necesidad de crear una instancia de la c lase. Los métodos estáticos no pueden usar variables de instancia.

Variables static: compartidas por todas las instancias de una clase.

FINAL indica que una vez inicializada el valor de la variable no puede cambiar. Generalmente se establecen como public para que puedan ser accedidas desde cualquier lugar de nuestro código, son STATIC para que no sea necesario crear una instancia de la clase para poder usarlas.

Se puede usar FINAL para variables de instancias, variables locales, parámetros de métodos y clases.

Variable: no puede cambiar su valor.

Método: no puede sobrescribirse.

Clase: no puede tener subclases.

CLASE ABSTRACTA similar a una clase concreta ya que también posee atributos y métodos, pero tiene una condición: al menos uno de sus métodos deber ser abstracto. Las clases abstractas no se pueden instanciar

Método abstracto: precedido por la palabra clave abstract, no tiene cuerpo y su encabezado termina con punto y coma. Se declara: digo que existe, lo define una clase que lo hereda.

Si un método se declara como abstrac se debe marcar la clase como tal: no puede haber métodos abstrac en una clase concreta.

Los métodos abstrac deben implementarse en las clases concretas (subclases) -> @OVERRIDE

CASTING conversión de una referencia de un tipo a otro.

INTERFAZ contrato entre una clase y una interfaz. Todos los métodos dentro de una interfaz son public y abstrac por defecto, de esta forma la clase que implemente dicha interfaz DEBE implementar los métodos.

COLLECTION API objeto que representa un grupo de objetos. El framework Collection es una arquitectura unificada para representar y manipular colecciones, lo que permite manipularlas independientemente de los detalles de la implementación ->

->Reduce los esfuerzos de programación al proveer estructuras de datos y algoritmos que no tenemos que escribir.

->Provee implementaciones de alta performance.

->Fomenta la reutilización del software al proporcionar una interfaz para colecciones y algoritmos para manipularlas.

INTERFAZ COLLECTION es la raíz de todas las interfaces y clases relacionadas con colecciones de elementos. Algunas colecciones permiten elementos duplicados mientras que otras no. Otras colecciones pueden tener los elementos ordenados mientras que en otras no existe orden alguno.

Es una interfaz porque:

Es la manera más genérica para representar un grupo de elementos.

Puede ser usada pasar colecciones de elementos o manipularlas de la manera más general.

Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes.

Se trata de métodos definidos por la interfaz que obligatoriamente han de implementar las subinterfaces o clases que hereden de ella.

LIST es la interfaz encargada de agrupar una colección de elementos en forma de lista, es decir uno atrás de otro. Acepta elementos duplicados y al igual que los arreglos, el primer elemento está en la posición 0.

ARRAYLIST basa la implementación de la lista en un array de tamaño variable. Un beneficio de usar esta implementación es que las operaciones de acceso a elementos, capacidad y saber si

es vacía se realizan de forma eficiente y rápida. Todo arraylist tiene una propiedad de capacidad, aunque cuando se añade un elemento esta capacidad puede incrementarse.

INTERFAZ COMPARABLE al implementarla en una clase estamos indicando que las instancias de dicha clase poseen un orden natural. El contrato de este método especifica que debe devolver: entero negativo si el objeto es menos al especificado por parámetro – cero si el objeto es igual – entero positivo si el objeto es mayor. declara el método compareTo() que compara su argumento implícito por el que se le pasa por parámetro, devolviendo -1, 0 ó 1 según el argumento sea anterior, igual o posterior al objeto.

COMPARATOR permite ordenar listas y colecciones cuyos objetos pertenecen a clases de cualquier tipo. La idea es parecida a la de Comparable, pero el usuario debe proporcionar la implementación de la interface. Esta interface declara los métodos equals(), que compara dos Comparators, y compare(), que devuelve -1, 0 ó 1 según el argumento sea anterior, igual o posterior al segundo.

INTERFAZ MAP nos permite representar una estructura de datos para almacenar pares "clave-valor". Para una clase solo tenemos un valor. Map tiene implementada por debajo toda la teoría de las estructuras de datos de árboles, por lo tanto, permite añadir, eliminar y modificar elementos de forma transparente. La clase funciona como un identificador único y no se admiten claves duplicadas.

INTERFAZ SET modela los conjuntos de la matemática y sus propiedades. Set hereda los métodos de Collection y agrega sus propias restricciones para prohibir el duplicado de elementos. Si tenemos un objeto que tienen las mismas características (equals) y el mismo hashCode que los objetos que ya se encuentran en el Set ->no se agrega a la colección.

HASHMAP colección de objetos, como los arrays, pero estos no tienen orden. Cada objeto se identifica mediante algún identificador y conviene que sea inmutable (final), de modo que no cambie en tiempo de ejecución. El nombre Hash hace referencia a una técnica de organización de archivos llamda hashing o "dispersión" en el cual se almacenan registros en una dirección del archivo que es generada por una función que se aplica a la clave del mismo. Los elementos que se insertan en un HashMap no tendrán un orden especifico. Permite una clave null.

HASHING 1- se utiliza el método hashCode ( ) para encontrar el bucket correspondiente, 2- se utiliza el método equals ( ) para buscar el valor correspondiente a la clave dada ->dos objetos tendrán el mismo identificador retornado por el método hashCode ( ).

HASHSET está implementación de Set almacena los elementos en una tabla hash. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash.

LINKEDHASHMAP similar a HashMap pero con la diferencia que mantiene una lista doblemente vinculada, además del array de baldes. La lista doblemente vinculada define el orden de iteración de los elementos.

METODO EQUALS ( ) Se debe redefinir el método para comparar el contenido de los objetos que queremos evaluar. Está estrechamente relacionado con la función hashCode ( ). Si sobrescribimos equals debemos sobrescribir también hashCode que debe cumplir que, si dos objetos son iguales, según la función equals, debe dar el mismo valor para ambos objetos ->

->Cuando NO: cada instancia de la clase es propiamente única – no hay necesidad para la clase de proveer una igualdad lógica – una superclase ya implemento este método y el mismo es apropiado para la clase – la clase es privada y estamos seguros que el método equals nunca será invocado.

->Cuando SI: cuando una clase tiene una igualdad lógica que va más allá de la identidad del objeto y una superclase no haya implementado este método.

METODO HASHCODE ( ) viene a complementar al método equals y sirve para comparar objetos de una forma más rápida en estructuras Hash ya que únicamente nos devuelve un numero entero ->

->Contrato: cuando este método es invocado en un objeto repetidas veces debe retornar el mismo valor consistentemente – si dos objetos son iguales de acuerdo al método equals, entonces hashCode debe retornar el mismo valor para ambos – si dos objetos son distintos de acuerdo al método equals, no es necesaria que hashCode produzca un valor distinto, sin embargo, esto puede mejorar la perfomance de un hash table.

METODOS EQUALS y HASHCODE siempre se debe hacer Override de hashCode cuando se hace Override de equals. Si dos objetos son iguales usando equals ( ), entonces la invocación a hashCode ( ) de ambos objetos debe retornar el mismo valor.

LINKEDLIST su implementación se basa en una lista doblemente vinculada de tamaño ilimitado. Al igual que arraylist también implementa la interfaz List. Su estructura está formada por Nodos, cada nodo contiene dos enlaces: uno a su nodo predecedor y otro a su nodo sucedor.

ARRAYLIST vs LINKEDLIST

A esta basada en una estructura de datos del tipo arreglo, mientras que L está basada en una lista doblemente vinculada.

VECTOR similar a un array que crece automáticamente cuando alcanza la capacidad inicial máxima. También puede reducir su tamaño, la capacidad siempre es al menos tan grande como el tamaño del vector.

VECTOR vs ARRAYLIST

V es sincronizada mientras que A no. Si no trabajamos en un entorno multihilos utilizar A. La estructura de ambos está basada en un array. Ambos pueden crecen y reducirse en forma dinámica, sin embargo, la forma en la que se redimensionan es diferente.

STACK representa una estructura LIFO, es una subclase de Vector, por lo tanto, su estructura también está basada en un array. Al igual que Vector, Stack es sincronizada.

QUEUE representa al tipo Cola, que es una lista en la que sus elementos se introducen únicamente por un extremo (fin de la cola) y se remueven por el extremo contrario (principio de la cola) FIFO

ITERATOR es un objeto que nos permite recorrer una lista y presentar por pantalla todos sus elementos . Dispone de dos métodos clave para realizar esta operación hasNext() y next().

BUCLE FOR EACH se parece mucho a un bucle for con la diferencia de que no hace falta una variable i de inicialización para recorrer una lista.