Paradigma Imperativo - Orientado a Objetos - Java

1. Teórico

a. Compare y contraste los conceptos de polimorfismo y sobrecarga de métodos en la programación orientada a objetos.

Comparación:

* Ambos tipos, cuando se invoquen, en tiempo de ejecución, se sabrá a quién se está referenciado.
* Ambos permiten la flexibilidad y extensibilidad del diseño de software.

Contraste:

* Son dos técnicas diferentes, ya que el polimorfismo al generar una relación entre clase padre y subclase, las firmas de los métodos se tienen que respetar, ya que es en el padre en donde se declara el comportamiento para delegar la responsabilidad a la clase hija que sabrá qué hacer.
* En la sobrecarga no se respeta la firma, sino que únicamente se respeta el nombre del método y se puede tener varias versiones de este en una entidad, para realizar una función que mejor coincida con sus parámetros.

Ejemplo Sobrecarga:

Public class Guerrero {

private int cantEnergia;

private int hp;

public List<Pocion> pocionesRegeneradorasDisponibles = new arrayList<Pocion>();

public List<Vendaje> VendasRegeneradorasDisponibles = new arrayList<Vendaje>();

public void curarse (int cantidadPociones) extends PocionException {

if(cantidadPociones > pocionesRegeneradorasDisponibles.size()){

throw new PocionException(“No tiene la cantidad suficiente de pociones para utilizar);

}

for (Pocion p : pocionesRegeneradorasDisponibles){

this.hp+=p.getPoderCuracion();

p.remove();

}

}

public void curarse (int cantidadPociones, int cantVendajes) extends PocionException , vendajeException, {

if(cantidadPociones > pocionesRegeneradorasDisponibles.size()){

throw new PocionException(“No tiene la cantidad suficiente de pociones para utilizar);

}

if(cantVendajes > VendasRegeneradorasDisponibles.size()){

throw new vendajeException(“No tiene la cantidad suficiente de vendajes para utilizar);

}

for(Pocion p : pocionesRegeneradorasDisponibles){

this.hp+=p.getPoderCuracion();

p.remove();

}

for(Vendaje v : VendasRegeneradorasDisponibles){

this.hp+=v.getPoderCuracion();

p.remove();

}

}

b. Explique en qué situaciones es apropiado utilizar uno u otro en el diseño de un sistema.

En el caso del polimorfismo, es adecuada usarla cuando tenemos una relación directa entre una clase padre y una subclase. El polimorfismo promueve las buenas prácticas, en donde vamos a permitir que, mediante un comportamiento heredado, se pueda instanciar específicamente en la subclase. El polimorfismo va de la mano del encapsulamiento (No se expondrá información que contiene una clase con las ajenas a ella misma. Ella sabrá cómo manejarse internamente)

La sobrecarga no necesariamente tiene que aparecer en un contexto de relación con otra clase. Lo que nos permite hacer es generar diferentes comportamientos en un mismo contexto, en base a lo que el método tenga por parámetro.

Ejemplo Polimorfismo:

public abstract class Transporte {

public abstract int calcularCombustibleViaje(int kmARealizar) extends ViajeException;

}

public class Camion extends Transporte {

private final static int CAMION\_LITROS\_POR\_KILOMETRO = 5;

@Override

public int calcularCombustibleViaje (int kmARealizar) extends ViajeException {

if(kmARealizar < 0){

throws new ViajeException(“El kilometraje no puede ser menor a 0”);

}

double combustibleViajeTotal = 0;

for (int i = 0; i < kmARealizar; ++i) {

combustibleViajeTotal+= kmARealizar \* CAMION\_LITROS\_POR\_KILOMETRO;

}

return combustiblePorKm;

}

}

2. Práctico

a. Modelar con UML un programa que modele una red social básica. Cree clases para representar usuarios, publicaciones y comentarios. Asegúrese de utilizar conceptos como herencia, polimorfismo y encapsulamiento en su diseño.

b. ¿Qué parte del sistema utiliza el polimorfismo? ¿Cómo?

c. Proporcione el código fuente de al menos dos métodos relevantes para el funcionamiento de la red social, como por ejemplo publicarMensaje y agregarComentario.

Paradigma Lógico - Prolog

3. Teórico

Explicar los distintos tipos de asignaciones y comparaciones existentes en Prolog. Proporcionar ejemplos concretos para ilustrar su funcionamiento.

Asignaciones:  
= (Asigna)  
  
A = 10

B = A

B va a ser 10  
  
is (compara o asigna. En caso de que el argumento de true se compara, o si es una operación arigmética, se asigna)

Comparación:

A = buenos\_aires

B = montevideo

A is B -> false

A = 10

B = 20

Resultado is (A+B)

Resultado termina siendo 30

Comparación:

== (Comparación)

A = buenos\_aires

B = Montevideo  
  
A == B -> Retonra false

=/= (Comparación not)

A = buenos\_aires

B = Montevideo  
  
A == B -> Retonra true

A = 10

B = 20

A > B :

< (compara si es menor) -> retorna true

> (Compara si es mayor) -> retorna false

=< (Compara si es menor igual) -> retorna true

>= (Compara si es mayor igual) -> retorna false

4. Práctico

a. Diseñar una base de conocimientos en Prolog con información sobre diferentes libros, incluyendo su título, autor, género y cantidad de páginas.

libro(pepito\_libro1, pepito,drama,100).

libro(pepito\_libro2,pepito2,suspenso,120).

libro(pepito\_libro3,pepito3, suspenso,130).

libro(pepito\_libro4,pepito4,terror,550).

libro(pepito\_libro5,pepito5,accion,198).

b. Luego, implementar reglas en Prolog que permitan al usuario realizar consultas y obtener información sobre

los libros, tales como:

1. obtener\_libros\_autor(Autor, Libros): Obtiene libros escritos por el autor especificado.

obtener\_libros\_autor(Autor, Libros):-

libro(Libros,Autor,\_,\_).

1. obtener\_libros\_con\_casi\_mas\_largos(Libros): Obtiene libros que tienen tantas páginas como el segundo más largo.

libro(pepito\_libro1, pepito,drama,100).

libro(pepito\_libro2,pepito2,suspenso,120).

libro(pepito\_libro3,pepito3, suspenso,130).

libro(pepito\_libro4,pepito4,terror,550).

libro(pepito\_libro5,pepito5,accion,198).

libro(pepito\_libro6,pepito5,accion,198).

libro(pepito\_libro7,pepito5,accion,198).

libro(pepito\_libro8,pepito5,accion,198).

obtener\_libros\_autor(Autor, Libros):-

libro(Libros,Autor,\_,\_).

prod\_cartesiano\_pag(X,Y):-

libro(\_,\_,\_,X),

libro(\_,\_,\_,Y).

seleccion\_menores(X,Y):-

prod\_cartesiano\_pag(X,Y),

X < Y.

proyeccion(X):-

seleccion\_menores(X,\_).

todos\_menos\_maximo(X):-

proyeccion(X).

obtener\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),

\+todos\_menos\_maximo(Cantpag).

todos\_menos\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),\+obtener\_mas\_largo(Libro,Cantpag).

prod\_cartesiano\_pag2(X,Y):-

todos\_menos\_mas\_largo(\_,X),

todos\_menos\_mas\_largo(\_,Y).

seleccion\_menores\_2(X,Y):-

prod\_cartesiano\_pag2(X,Y),

X < Y.

proyeccion2(X):-

seleccion\_menores\_2(X,\_).

todos\_menos\_maximo2(X):-

proyeccion2(X).

todos\_menos\_segundo\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),

\+todos\_menos\_maximo2(Cantpag),

\+obtener\_mas\_largo(Libro,Cantpag).

obtener\_libros\_con\_casi\_mas\_largos(Libros):-

todos\_menos\_segundo\_mas\_largo(Libros,\_). libro(pepito\_libro1, pepito,drama,100).

libro(pepito\_libro2,pepito2,suspenso,120).

libro(pepito\_libro3,pepito3, suspenso,130).

libro(pepito\_libro4,pepito4,terror,550).

libro(pepito\_libro5,pepito5,accion,198).

libro(pepito\_libro6,pepito5,accion,198).

libro(pepito\_libro7,pepito5,accion,198).

libro(pepito\_libro8,pepito5,accion,198).

obtener\_libros\_autor(Autor, Libros):-

libro(Libros,Autor,\_,\_).

prod\_cartesiano\_pag(X,Y):-

libro(\_,\_,\_,X),

libro(\_,\_,\_,Y).

seleccion\_menores(X,Y):-

prod\_cartesiano\_pag(X,Y),

X < Y.

proyeccion(X):-

seleccion\_menores(X,\_).

todos\_menos\_maximo(X):-

proyeccion(X).

obtener\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),

\+todos\_menos\_maximo(Cantpag).

todos\_menos\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),\+obtener\_mas\_largo(Libro,Cantpag).

prod\_cartesiano\_pag2(X,Y):-

todos\_menos\_mas\_largo(\_,X),

todos\_menos\_mas\_largo(\_,Y).

seleccion\_menores\_2(X,Y):-

prod\_cartesiano\_pag2(X,Y),

X < Y.

proyeccion2(X):-

seleccion\_menores\_2(X,\_).

todos\_menos\_maximo2(X):-

proyeccion2(X).

todos\_menos\_segundo\_mas\_largo(Libro,Cantpag):-

libro(Libro,\_,\_,Cantpag),

\+todos\_menos\_maximo2(Cantpag),

\+obtener\_mas\_largo(Libro,Cantpag).

obtener\_libros\_con\_casi\_mas\_largos(Libros):-

todos\_menos\_segundo\_mas\_largo(Libros,\_).

Paradigma Funcional - Haskell

5. Teórico

Explique el concepto de listas por comprensión en Haskell y cómo se utilizan para construir listas de manera concisa y elegante. Describa cómo se definen las listas por comprensión y cómo se pueden aplicar filtros y transformaciones a través de ellas. Proporcione ejemplos concretos para ilustrar el uso de listas por comprensión.

Las listas por comprensión permiten definir listas por una notación, una sintaxis, de manera comprensiva y definir por una guarda (expresión booleana) elementos locales.

Ejemplo:

Expresión que filtra los números pares del 1 al 5

Def local | generador | guarda

[ x | x <- [1..5], even x]

Estas listas, resuelve lo mismo que se puede hacer con un map.

6. Práctico

Implementar una función en Haskell que tome un número y devuelva una función que sume ese número a cualquier otro número. Utilice el currying para lograr esta funcionalidad.

Ejemplo:

sumar5 = sumarN 5

sumar5 10 -- retorna 15

incrementar:: Int -> Int

incrementar x = x + 1

twice :: (a -> a) -> a -> a

twice f x = f (f x)

main = do

print (twice incrementar 0)