**Documentación sobre decisiones de diseño**

***Entrega 1***

1. **Categorías, Colores y Telas**

Optamos por diseñarlas con Enums porque cada clase solamente va a poder tener un conjunto limitado de valores, y de ésta forma el programa no te deja insertar cualquier otro valor no definido. Ejemplo:

Supongamos que a la categoría la diseño como un String. Yo voy a poder asignarle a esa variable la cadena “parte superior”, un compañero puede asignarle “torso”, otro compañero puede asignarle “bleh”, incluso puedo asignarle “parte suprior”. Si se produce un error grosero así en el tipeo, cuando quiera conseguir las prendas correspondientes a la parte superior, esa última prenda nunca va a ser tomada en cuenta.

1. **Tipos de Prenda**

Diseñamos los tipos de prenda con herencia, con una clase abstracta Prenda y distintos tipos que serán los hijos (Pantalón, remera, zapatos, y más). ¿Por qué diseñarlo de ésta forma y no con un “String tipo”? La primera razón es que el tipo de tela tenía que ser consistente con el tipo de prenda que se quería construir, por lo tanto, los distintos tipos de prenda iban a tener un conjunto de telas con las cuales no debían poder construirse. Si esto fuera a desarrollarse con un String, en el constructor de la Prenda tendría que haber puesto :

If(tipo==”unTipoEspecífico”){

inicializarTelasInconsistentes();

}

Ese código debería escribirse en el constructor de la prenda, por cada tipo diferente de prenda que tengamos en nuestro sistema, por lo tanto, un inocente constructor tendrá de ésta forma tantos if como tipos de prenda haya. Además está el tema de los errores, ya que si quiero por ejemplo crear una remera de cuero, el sistema no deberá permitirlo. Entonces el código a insertarse debería ser:

If(tipo==”unTipoEspecífico”){

inicializarTelasInconsistentes();

if(prendaInconsistente()){ throw new ExcepcionTipoDePrenda;

}

Ahora pasamos a tener una secuencia de 2 “if” por cada tipo, estaríamos agregando una enorme cantidad de código a un constructor que además de construir una prenda debe ahora chequear por tipos y tirar errores por cada tipo. Entonces, ¿no puede hacer eso el tipo de prenda? Entonces teniendo clases como tipo de prenda, cada constructor de esas subclases inicializa sus Telas inconsistentes (telas con las que nunca podrá ser construida) y chequea si la Tela de la prenda se encuentra en esa Lista, y de ser así, rompe.

1. **Guardarropa**

Optamos por diseñar las prendas de un guardarropa como una Lista de prendas, aceptando cualquier subtipo.

1. **Sugerencia**

Para la implementación de éste algoritmo utilizamos la combinatoria entre elementos de vectores:

Hicimos un filtro por cada categoría de prenda, quedando así 4 vectores. Luego combinamos las posiciones de todos los vectores entre sí, y así íbamos obteniendo cada combinación llamada Atuendo. Ejemplo:

V1:[ V1a,V1b,V1c]

V2:[ V2a,V2b,V2c]

V3:[ V3a,V3b,V3c]

V4:[ V4a,V4b]

El primer Atuendo sería: [V1a,V2a,V3a,V4a]

Así obtenemos una lista de atuendos, y decidimos por el momento contemplar 2 opciones: devolver la lista entera o devolver la primera combinación.

1. **Atuendo**

Diseñamos esto como una abstracción más en nuestro sistema, para tener de forma organizada las prendas de las diferentes categorías, y además le agregamos el método “compararConOtroAtuendo” para que se fije si tiene los mismos atributos que otra combinación. Esto nos sirvió para testear: a mano creamos un Atuendo con la combinación de prendas que debería obtener al aplicar el algoritmo, y lo comparamos con el Atuendo obtenido al aplicar el mismo.