**Aclaraciones**:

– Resuelvo el primer y segundo requerimiento con una clase guardarropas que contenga prendas de su usuario duenio.El usuario ahora tiene a sus prendas a travez de algun guardarropas.Cada uno de estos es de un ambiente , el cual se corresponde con un criterio de separacion de la ropa.Modelo el mismo como un enum porque no creo que puedan inventarse muchos criterios.Al menos temporalmente es viable modelarlo como un enum.

El guardarropas lo hice una clase para poder compartirlo facilmente emtre usuarios.Si fuera algun tipo de colección , habria que duplicarla en cada ususario con la que se comparte dado que una colección no es una entidad y , por eso , no se puede referenciar.

Supongo que una prenda solo puede estar en un guardarropas porque es lo logico en la vida real (aunque tener referencias a una misma instancia de Prenda en distintas instancias de Guardarropas sea posible).

Basandome en esa suposicion , hago que la prenda tambien conosca su ambiente para poder validar , en el futuro , que en un guardarropa de cierto ambiente solo haya prendas de ese ambiente.

– Resuelvo el tercer y cuarto requerimiento con una clase propuesta y un metodo en el usuario que la genera.Las mismas deben estar cosificadas en una clase para cumplir el quinto requerimiento.

–Resuelvo el quinto requerimiento con dos colecciones en el usuario , una para las propuestas que recibe y otra para las que acepta.Y dos metodos , uno que elimna una propuesta recibida y otro que la mueve a la colección de aceptadas.Y un metodo mas que muestra las propuestas hechas sobre un cierto guardarropas.

– Resuelvo el sexto requerimiento con un metodo que elimina propuestas de la colección de aceptadas.

**Pseudocodigo JAVA**:

**class Usuario{**

… (atributos de entregas anteriores)

[Guardarropa] guardarropas;

[Propuesta] propuestasPendientes;

[Propuesta] propuestasAceptadas;

**void ProponerCambio(Usuario usuario , [Prenda] prendas , Guardarropas guardarropas , TipoPropuesta tipo){**

usuario.getPropuestasPendientes().add(new Propuesta(prendas , guardarropas , tipo));

**}**

**[Propuesta] getPropuestasPendientes(){** return this.propuestasPendientes; **}**

**[Propuesta] getPropuestasAceptadas(){**return this.propuestasAceptadas; **}**

**[Propuesta] getPropuestasDeUnGuardarropas(Guardarropas guardarropas){**

return this.propuestasPendientes.filter(p → p.getGuardarropas() == guardarropas);

**}**

**void DeshacerPropuestaAceptada(Propuesta propuesta){**

this.propuestasAceptadas.remove(propuesta);

**}**

**void RechazarPropuesta(Propuesta propuesta){**

this.propuestasPendientes.remove(propuesta);

**}**

**void AceptarPropuesta(Propuesta propuesta){**

if(this.propuestasPendientes.contains(propuesta)){

this.propuestasAceptadas.add(propuesta);

this.propuestasPendientes.remove(propuesta);

} else {;}

**}**

… (metodos entregas anteriores)

**}**

**class Guardarropas{**

[Prenda] prendas;

AmbienteRopa ambiente;

**}**

**class Propuesta{**

[Prenda] prendas;

Guardarropas guardarropas;

TipoPropuesta tipo;

**Propuesta([Prenda] prendas , Guardarropas guardarropas , TipoPropuesta tipo ){**

this.guardarropas = guardarropas;

this.tipo = tipo;

prendas.forEach( p → this.prendas.add(p) );

**}**

**Guardarropas getGuardarropas(){** return this.guardarropas;**}**

**}**

**class Prenda{**

AmbienteRopa ambiente;

… (atributos de entregas anteriores)

… (constructor y metodos de entregas anteriores)

**}**