Tecnológico de Costa Rica

Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández

II Semestre, 2018 II Examen Parcial

Estudiantes:

Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez

Equipo de Trabajo	Punto A 25 puntos	Punto B 60 puntos (30 c/u)	
1. Andrés Brenes Maleaño	Caso 1	Caso 3	Caso 4
2. Axel Fernández Jiménez	Caso 2	Caso 5	Caso 6
Puntos Obtenidos	Punto A	Punto B.1	Punto B.2
Total Puntos		**Nota**	

Tecnológico de Costa Rica

Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández

II Semestre, 2018 II Examen Parcial

Estudiantes:

Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez

Caso 1

En el mapa de la Tierra Media existen múltiples razas: Elfos, Orcos, Enanos, Maiars, entre otros. Todas estas razas siempre están en guerra.

Ahora ha aparecido un herrero como jamás ha existido en la historia, y ofrece el servicio de proveer armas para cualquiera de las razas pues conoce a la perfección la forma de fabricar cada tipo de ellas para la raza particular.

Dentro de las armas que puede fabricar se mencionan:

Dagas élficas	Armas Orcos	Hacha Enanos	Garrote Sauron
			

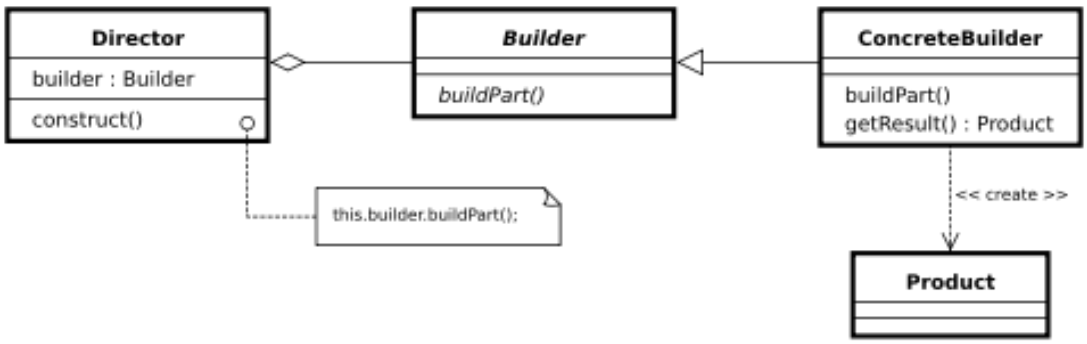
- Justifique el patrón que se ajusta al caso escogido
    - El patrón elegido para solucionar este problema es el **builder**. En este caso poseemos diferentes razas que desean construir diferentes armas, ya que estas siempre se encuentra en una guerra, y nos encontramos con un herrero, que tiene la tarea de crear estas armas, por lo que él sería un constructor. El **builder**, se hace presente cuando las distintas armas, se ven, desde fuera, como un arma abstracta, y se le da forma, dependiendo de la raza, por lo que cada creación de arma, es distinta y es necesario tener un proceso diferente para cada una, haciendo bastante visible el patrón **builder**.
- Diagrama de UML original del patrón seleccionado.

Tecnológico de Costa Rica | Ingeniería en Computación

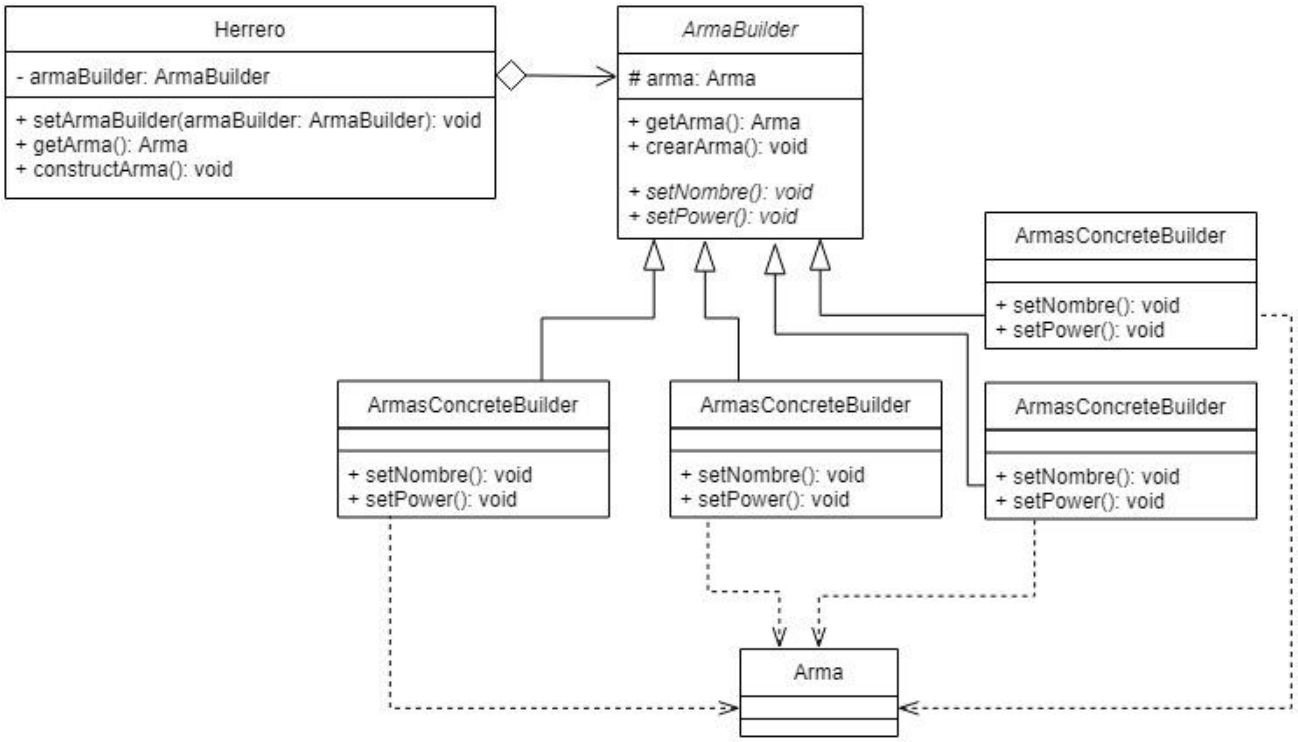
Prof: Ing. Ericka Solano Fernández | II Semestre, 2018 II Examen Parcial

Estudiantes: Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez



- Modele el caso de acuerdo con el patrón seleccionado.



- Explicación de la responsabilidad que tendrá cada uno de los componentes en el modelo propuesto.
    - Herrero: En el caso, este tomaría el caso de Director, ya que es quien se encarga de construir cada arma que existe; es el experto que sabe cómo hacerlo.

- ArmaBuilder: Es una clase abstracta que tiene la función de Builder en el caso actual. Se utiliza para que el herrero pueda construir cada arma como si fuera una misma.
- DagasElficas, ArmasOrcos,HachaEnanos,GarroteSauron: Son los diferentes Concrete Builders que se encargan de definir cómo es construida el arma para el cual son creados
- Arma: Es el producto final, que dependiendo del tipo de arma fabricado, tiene distinto valor de poder.

Tecnológico de Costa Rica

Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández

II Semestre, 2018 II Examen Parcial

Estudiantes:

Andrés Ottón Brenes Maleaño

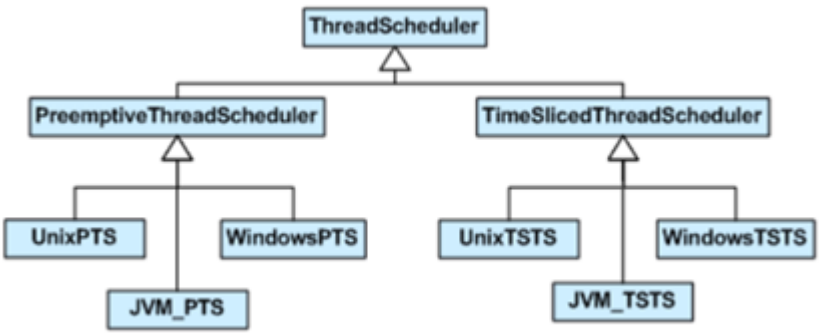
Axel Alejandro Fernández Jiménez

Caso 3

Los sistemas operativos dentro de sus funciones deben planificar la forma en que atenderán los procesos que van solicitando sus servicios. Suponga que dos sistemas operativos distintos podrían definir la atención de sus procesos en cualquiera de dos esquemas que se han definido: uno preventivo (Preemptive Thread Scheduler) y otro por secciones de tiempo (TimeSliced Thread Scheduler).

Si en determinado momento se cuenta con un nuevo sistema operativo u otro esquema de atención, se desea que cualquiera de estos sistemas operativos pueda incorporar cualquiera de sus esquemas de manejo de procesos. Es decir, que se puedan realizar distintas combinaciones sistemas operativos - esquemas de atención de procesos.

Lo anterior se puede mostrar de manera visual de la siguiente forma:



¿Cuál patrón podría ayudar a mejorar este esquema de modo que se pueda satisfacer cualquier combinación de sistema operativo con esquema de manejo de procesos, inclusive si surgieran otros en cualquier momento?

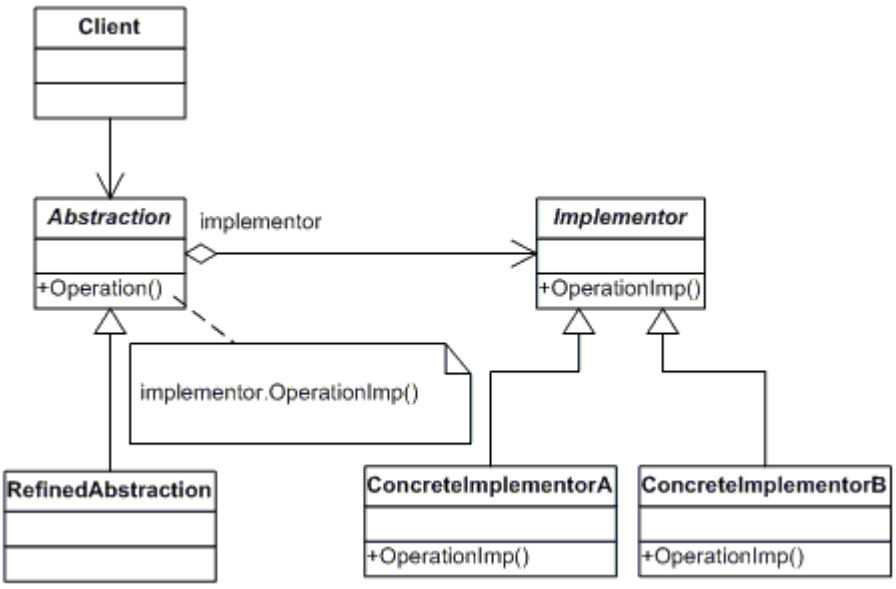
Tecnológico de Costa Rica — Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández — II Semestre, 2018 II Examen Parcial

Estudiantes: Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez

- Justifique el patrón que se ajusta al caso escogido
    - El patrón elegido para solventar este problema es el **bridge**. El cliente puede solicitar dos diferentes procesos, que son el *PreemtiveThreadScheduler* y el *TimeSlicedThread*, y desde cualquier sistema operativo. Por lo que, un **bridge**, desconecta completamente que el sistema operativo debe tener un proceso dedicado a cada solicitud del cliente, sino que, se tiene una interface que posea los distintos procedimientos deseados y de esta manera, los diferentes sistemas operativos puedan ejecutar el proceso solicitado, empacando la respuesta en una misma ejecución.
- Diagrama de UML original del patrón seleccionado.



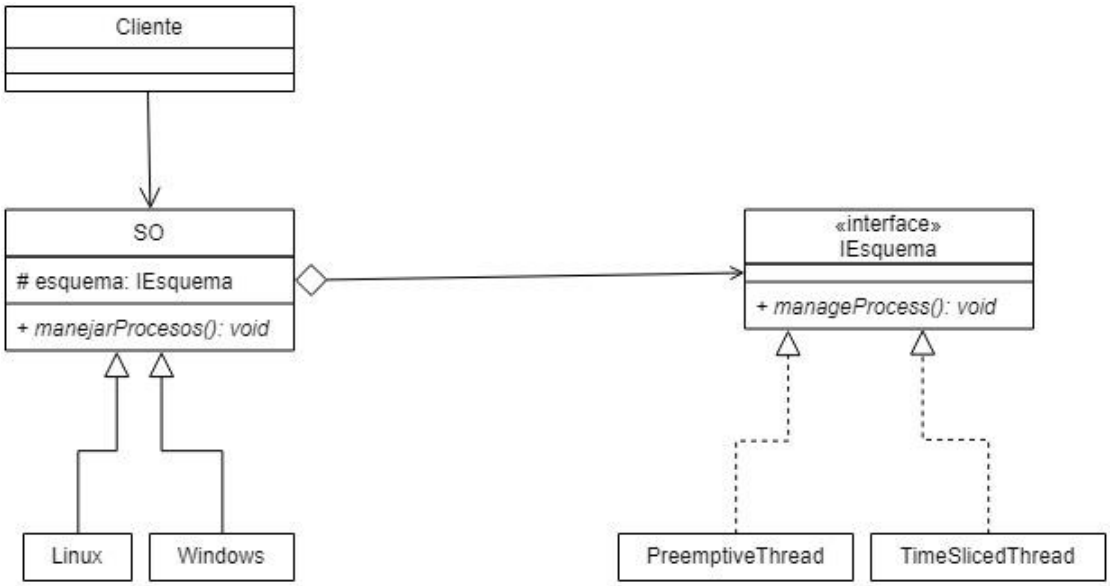
Tecnológico de Costa Rica | Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández | II Semestre, 2018 II Examen Parcial

Estudiantes: Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez

- Modele el caso de acuerdo con el patrón seleccionado.



- Explicación de la responsabilidad que tendrá cada uno de los componentes en el modelo propuesto.
    - SistemaOperativo: es una clase abstracta que se encarga de definir las operaciones de los Sistemas Operativos. En el patrón de diseño, sería Abstraction.
    - Linux, Windows: son las clases concretas que extienden Sistema Operativo. En el patrón, serían RedefinedAbstraction.
    - IEsquema: Interfaz que se encarga de definir la implementación del método manageProcess. En el patrón sería Implementer
    - PreemptiveThread, TimeSlicedThread: Se encargan de definir la manera en que se manejan los procesos. En el patrón serían los ConcreteImplementator.

Tecnológico de Costa Rica

Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández

II Semestre, 2018 II Examen Parcial

Estudiantes:

Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez

Caso 6

Cuando nos enfrentamos al desarrollo de aplicaciones de gran dimensión, es muy frecuente que determinadas clases que tienen como objetivo realizar tareas repetitivas en pasos sucesivos vayan creciendo y creciendo debido a nuevos requerimientos.

Suponga que existe un determinado formulario que maneja información de usuarios con sus datos básicos: nombre, apellidos, edad, contraseña de acceso, datos de localización, los cuales pueden ser de cualquiera de los tipos: dirección física, apartado postal, correo electrónico, número de celular.

El formato de dónde proviene la información es irrelevante.

Al ingresar los datos de un nuevo usuario se desea realizar una revisión a través de una mecanismo de validación que revise todo el formulario de modo que devuelva los posibles errores encontrados en todos y cada uno de los campos suministrados por el usuario, en lugar de asociar mecanismos de validación de formatos a cada uno de los campos.

Por lo que muchos campos podrían considerarse obligatorios y no se debería permitir que sean vacíos.

Adicionalmente, podría requerirse asociarse un validador adicional para el campo de la edad que revise si cumple con la mayoría de edad o que la edad esté en un rango determinado.

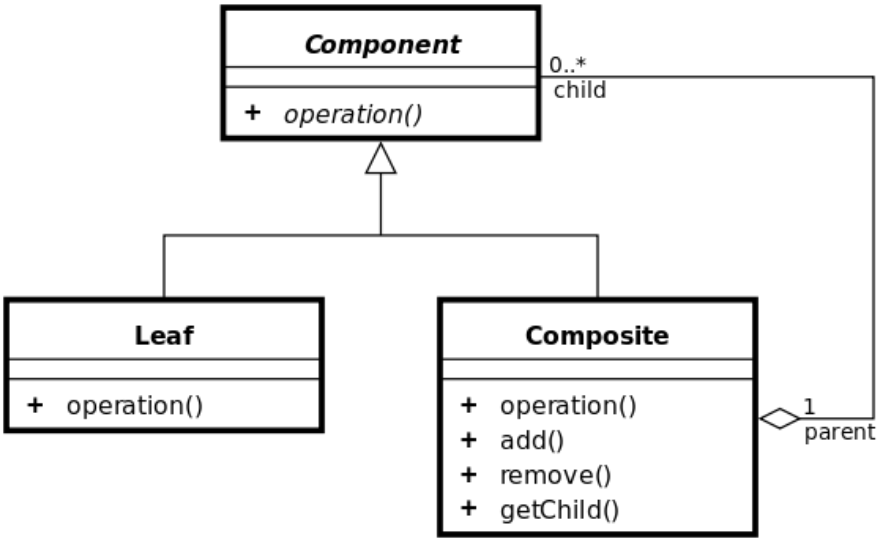
Eventualmente se podría requerir un dato de localización tenga el formato adecuado: el correo electrónico, el apartado postal o una dirección dada como Provincia-Cantón-Distrito.

Finalmente se podría requerir que una contraseña cumpla con ciertas reglas.

Entonces la idea es poder asociar entre 0 y N validadores a un campo de captura de datos que forme parte del formulario o pantalla de captura de datos y que cuando ésta se envía se lleva a cabo un proceso de revisión de

validadores que aportan cada uno de ellos el resultado de los validadores asociados a cada campo en particular.

- Justifique el patrón que se ajusta al caso escogido
    - El patrón elegido para solventar este problema es el **composite**. En un sistema que posee un usuario con información relevante que es necesario realizar una comprobación cada vez que sea solicitado, se procede a crear una serie de validaciones para verificar que todo esté en orden con la persona que activa el proceso. De esta manera, se puede representar un composite, debido a que es necesario realizar validaciones de diferentes características del usuario, y estas pueden ampliarse por lo que la creación de validaciones nuevas dentro del sistema es posible. A la vez, es necesario poder acceder a una validación que se solicite, por lo que el patrón de diseño **composite** es más que claro para llevar a cabo la implementación de este problema.
- Diagrama de UML original del patrón seleccionado.



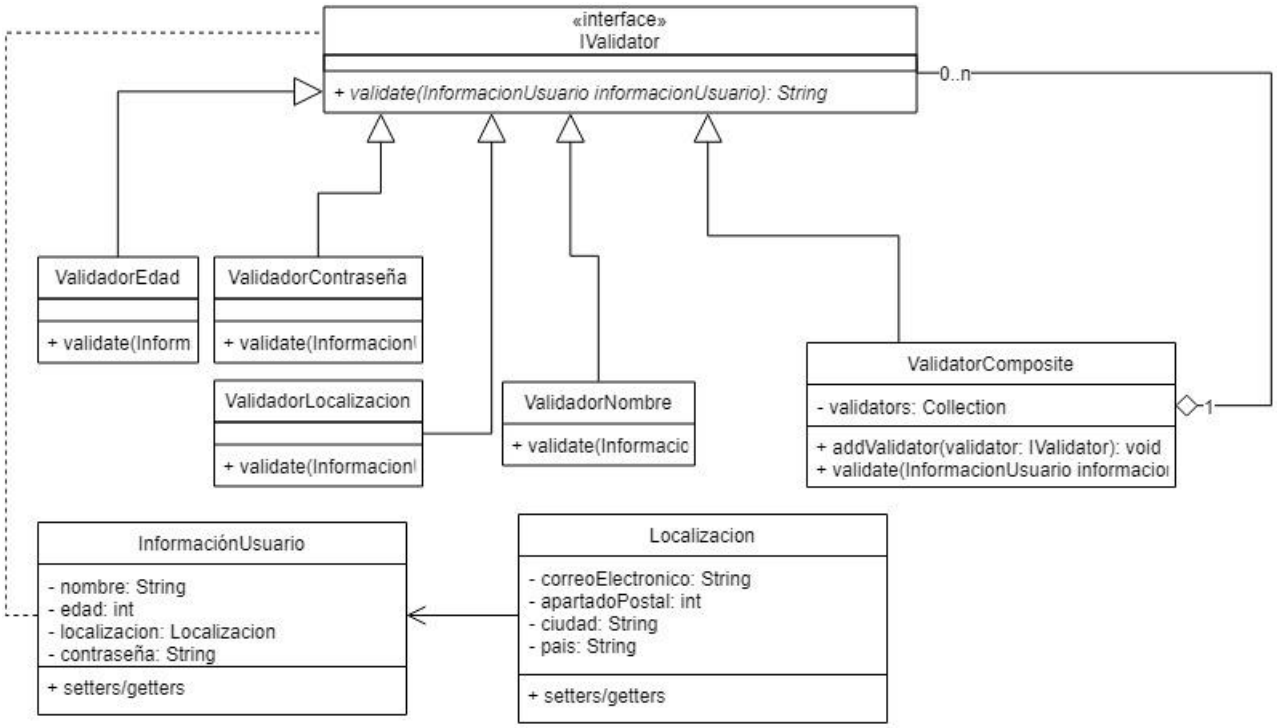
- Modele el caso de acuerdo con el patrón seleccionado.

Tecnológico de Costa Rica | Ingeniería en Computación

Prof: Ing. Ericka Solano Fernández | II Semestre, 2018 II Examen Parcial

Estudiantes: Andrés Ottón Brenes Maleaño

Axel Alejandro Fernández Jiménez



- Explicación de la responsabilidad que tendrá cada uno de los componentes en el modelo propuesto.
    - IValidator: Interfaz que se encarga de definir la implementación de validar. Toma el papel de Component.
    - ValidatorComposite: Se encarga de tener una lista con los diferentes validadores que se pueden aplicar a la información del usuario. Como su nombre indica, tiene el rol de Composite.
    - ValidadorEdad, ValidadorNombre, ValidadorContraseña, ValidadorLocalización: Se encargan de validar cada uno de los campos que el cliente les manda. Son la implementación concreta de IValidator de manera individual; se comportan como la clase Leaf.
    - InformacionUsario, Localizacion: Clase aparte que son validadas por medio de la interface IValidator.