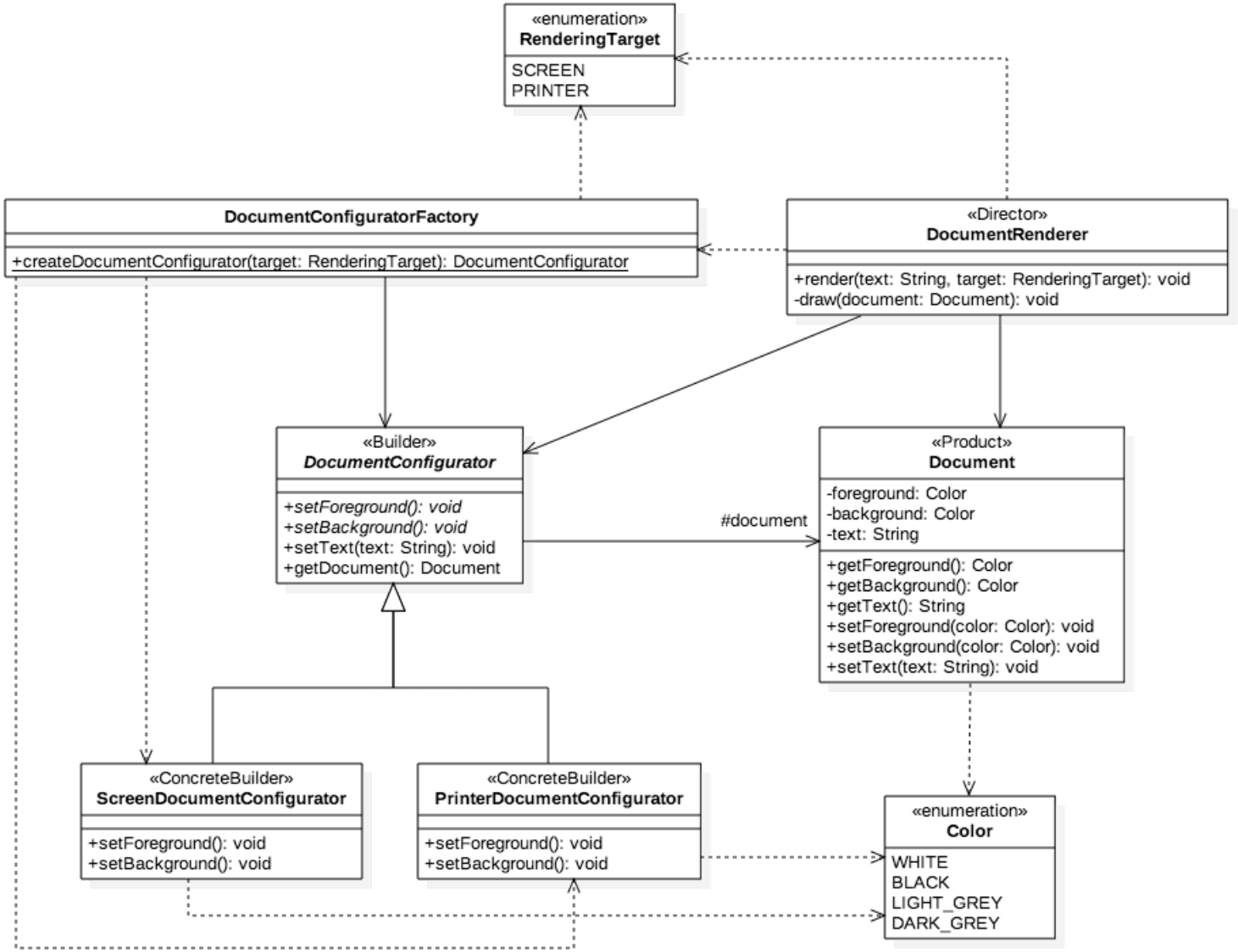
Diseño de Sistemas Software – Solución examen junio 2018

Ejercicio 1



Diseño de Sistemas Software – Solución examen junio 2018

- Es importante reflejar todas las dependencias entre las clases del diagrama.
- Los métodos estáticos se indican con subrayado.
- Las clases y métodos abstractos se indican en cursivas. Al dibujar el diagrama a mano se puede sustituir las cursivas por el estereotipo <<abstract>>.

Ejercicio 2

El patrón empleado en el código es **Builder**. La clase DocumentRenderer (Director) dirige el proceso de construcción, indicando a DocumentConfigurator (Builder) los pasos a realizar para obtener una instancia de Document (Product):

```
documentConfigurator.setForeground();
documentConfigurator.setBackground();
documentConfigurator.setText(text);
Document document = documentConfigurator.getDocument();
```

La clase DocumentConfigurator es abstracta. Las clases ScreenDocumentConfigurator y PrinterDocumentConfigurator (ConcreteBuilders) conocen los detalles de implementación de los pasos setForeground() y setBackground().

Todos los roles están indicados en el diagrama mediante estereotipos.

La factoría empleada en el código es solamente una clase auxiliar para obtener una instancia de uno de los dos ConcreteBuilder. Se trata de una factoría sencilla, no de los patrones GOF Abstract Factory o Factory Method. Estos dos patrones se basan en el uso de una factoría abstracta que deben extender otras factorías derivadas, que son las que se encargan de crear instancias de distintos tipos de productos.

Ejercicio 3

El enunciado indica dos restricciones clave para la solución del problema:

- El texto modificado no se debe almacenar en el documento → las transformaciones no se deben hacer antes de llamar a setText().
- Las transformaciones no se deben hacer en las clases Document ni DocumentRenderer → la clase document debe devolver el texto original al llamar a getText(), pero DocumentRenderer debe obtener el texto ya modificado.

El patrón más adecuado para solucionar este problema es **Decorator** aplicado sobre la clase Document. De esta forma se definen clases que mantienen el mismo interfaz que Document, por lo que DocumentRenderer no se ve afectado por el cambio, pero que al mismo tiempo permiten extender su funcionalidad añadiendo todas las transformaciones necesarias.

Las clases ScreenDocumentConfigurator y PrinterDocumentConfigurator sobreescriben el método getDocument() para devolver el documento decorado con los decoradores que realizan las transformaciones que corresponden en cada caso.

Diseño de Sistemas Software – Solución examen junio 2018