Justificaciones de Diseño

Para la clase EquipoDeCompeticion se decidió tener una lista de participantes en lugar de una posible clase abstracta. De esta manera:

* Se evita el acoplamiento entre las clases hijas y padre.
* Se evita que no sean extensibles en tiempo de ejecución (y se logra que sí lo sean al ser clases concretas).
* Se evita forzar la herencia dado que no existe diferencia de comportamiento entre las clases hijas.

Esto mismo es válido para el modelado de vehículos.

Al separar la clase Participante de Persona, se contempla el caso en que una persona sea corredor en una competencia y técnico en otra.

Se modeló el TipoDocumento como un enum en vez de clase concreta dado que no es necesario que sea extensible en tiempo de ejecución y se evita el problema de inconsistencia de datos.

Se modeló la Nacionalidad como clase concreta para no tener un enum con 250 entradas aproximadamente. De todas maneras, esto último hubiera sido válido dado que no es necesario que sea extensible en tiempo de ejecución y se evita el problema de inconsistencia de datos.

Se modelaron las clases Marca, Modelo y Rodado como clases concretas para que sean extensibles en tiempo de ejecución.

El enunciado no aclara cuáles deberían ser los parámetros que la API del Renaper necesita para realizar la verificación por lo que puse los que me parecían que tenían más sentido (tampoco se sabe qué tipo de dato devuelve). Por este motivo, se dejó a la clase APIRenaper vacía cuando realmente debería tener por lo menos un método.

Para la verificación de identidad, decidí utilizar el patrón Adapter para bajar el acoplamiento con la APIRenaper. De esta manera, si en un futuro la API quedase deprecada, sería fácilmente reemplazable por otra clase que implemente la interface IAdapterRenaper. Además, el patrón nos brinda:

* Mayor mantenibilidad debido al nulo acoplamiento entre el VerificadorDeIdentidad y la APIRenaper.
* Mayor cohesión de la clase VerificadorDeIdentidad debido a la delegación de cierto comportamiento en el adapter.
* Mayor facilidad de testeo ya que se podría implementar un mock de AdapterConcretoRenaper.

También soluciona/evita de forma directa los siguientes code smells:

* Herencia rechazada.
* Clases alternativas con diferentes interfaces.
* Clase de biblioteca incompleta.

Esto mismo es válido para la verificación de antecedentes penales y la notificación de mensajes.

En caso de ser necesario y para lograr una mayor cohesión y menor acoplamiento, podrían agregarse clases Mappers para la conversión de tipos de dato al momento de comunicarse con las APIs ya sea para el envío o recepción de datos.

La clase EntidadOrganizadora fue modelada como clase concreta dado que el enunciado no aclara si debería ser extensible en tiempo de ejecución.

La clase TipoDeActividad fue modelada como clase concreta dado que el enunciado aclara que pueden incorporarse nuevos tipos de actividad en el futuro, por lo que debería ser extensible en tiempo de ejecución. De esta manera, se logra el dinamismo pedido.

Se modeló la clase SolicitudDeInscripcion para tener una mayor trazabilidad de las inscripciones.

Para la inscripción a una competencia, la clase Inscriptor recibe por inyección de dependencia a las clases VerificadorDeAntecedentesPenales, VerificadorDeAntecedentesDeTransito, Notificador, EquipoDeCompeticion y Competencia, logrando una mayor cohesión y menor acoplamiento con estas clases (además de evitar una clase Dios). Esta clase llama al builder crearSolicitudDeInscripcion() que es un método estático de la clase SolicitudDeInscripcion que se encarga de su instanciación. Además, el Instanciador delega la verificacion de antecedentes llamando a las clases VerificadorDeAntecedentesPenales y VerificadorDeAntecedentesDeTransito. En caso de tener antecedentes penales, el VerificadorDeAntecedentesPenales setea el estadoSolicitud de la SolicitudDeInscripcion en RECHAZADA, la fechaYHoraResultadoInscripcion y setea en false el atributo antecedentesPenalesVerificados de la clase SolicitudDeInscripcion. El VerificadorDeAntecedentesDeTransito realiza acciones muy similares a las recién descriptas.

Tanto el VerificadorDeAntecedentesPenales como VerificadorDeAntecedentesDeTransito son cronTasks, ejecutan de manera asincrónica una vez al día todos los días a la noche. El VerificadorDeAntecedentesPenales utiliza un mecanismo push based (será notificado cuando la API REST responda) mientras que el VerificadorDeAntecedentesDeTransito utiliza un mecanismo pull based (ejecutará una vez al día todos los días hasta que el registro que le interesa aparezca en la base de datos compartida).

Se modeló la clase Mensaje para tener una mayor trazabilidad de los mensajes enviados.

Para mayor cohesión y menor acoplamiento, se modeló la clase GeneradorDeMensaje. De esta manera, también se evita que el Notificador quede como una clase Dios.

Arquitectura (Punto 1.1)

Para el contexto y requerimientos previstos, sugeriría considerar los atributos de calidad: interoperabilidad y disponibilidad.

Interoperabilidad dado que es extremadamente prioritario que el sistema pueda intercambiar información y utilizar la información intercambiada con la API REST.

Disponibilidad dado que es primordial que el sistema se encuentre operativo y accesible para su uso, principalmente en días de alto tráfico como pueden ser los cercanos a fechas límite.