Universidad ORT Uruguay
Facultad de Ingeniería Bernard-Wand Polak

Obligatorio 1: Diseño de Aplicaciones 1

Mateo Giraz - 241195
Renzo Testorelli - 251459
Santiago Cabrera - 266191

Repositorio GitHub

Tutores:
Guillermo Areosa y Bruno Balduccio

2023

Índice

Organización del trabajo

SCM

Organizamos el trabajo en un repositorio remoto GIT de forma que pudiéramos trabajar de forma autónoma. Hicimos empleo de Git Flow para gestionar nuestro trabajo, creando ramas para cada función y retrasando su fusión en develop hasta que su desarrollo esté completo. De esta forma mantenemos develop como una rama siempre funcional. A su vez existe la rama main, que está protegida y solo tiene utilidad cuando se fusiona el proyecto terminado.

Tablero

Cada una de estas nuevas ramas está vinculada a un ticket de Jira, en cuyo tablero los integrantes del equipo pueden mover las incidencias entre las categorías *TO DO*, *IN PROGRESS, WAITING FOR PR APPROVAL, DONE,* de tal forma que el resto de los desarrolladores conoce el estado de las tareas.

Pull request

Cuando termina el desarrollo de una funcionalidad, el desarrollador realiza una pull request para fusionar su trabajo en la rama develop. De esta manera, otro integrante del equipo puede leer su trabajo y aceptarlo en caso de que lo encuentre correcto, o hacer un comentario si desea que se realice algún cambio.

Estándares de trabajo

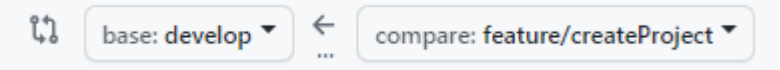
Commits

Desarrollamos nuestra aplicación con la metodología **TDD**, por tanto nuestros commits normalmente refieren a los ciclos del desarrollo basado en tests. Estos son los commits correspondientes en función de la etapa del ciclo.

Etapa roja -> *“red - <testFunctionName>”*
Etapa verde -> *“green - <testFunctionName>”*
Etapa de refactorizado -> *“refactor - <testFunctionName>”*

Comenzar una nueva funcionalidad

Para empezar una nueva funcionalidad, tomamos un ticket y creamos una nueva rama con el nombre ***“feature/<ticketName>”***, cuando está terminada, creamos una **pull request** con base develop.



Creación de pull request

Para crear las pull request, seguimos el siguiente estándar:

- ## <**ticketName**>
- Ticket: <**ticketLink**>
- Descripción (opcional)
- Captura de pantalla de la funcionalidad (opcional)
- Captura de pantalla del ticket

Aprobación de pull request

TÍTULO: “MERGE feature/<**ticketName**> INTO develop”

DESCRIPCIÓN: “<**ticketName**>”

Instalación

Se entrega, en el último commit del obligatorio, la documentación, el release con la aplicación compilada, el código fuente de la aplicación, y el directorio de bases de datos, que incluye tanto para la base de datos principal como de backup, los archivos .bak y los scripts de creación. El ejecutable se encuentra dentro de la carpeta release. Para iniciar la aplicación, primero es necesario restaurar la base de datos, para ello se encuentran los archivos .bak con y sin datos, en las siguientes rutas: *db/Schema&Data/Backup.bak* y *db/SchemaOnly/Backup.bak* respectivamente.

Descripción general del sistema

Nuestra aplicación fue desarrollada haciendo uso de la metodología **TDD**, causando que el código siempre esté respaldado por tests, facilitándonos así la modificación y mantenibilidad del mismo. A su vez, TDD nos permite minimizar la cantidad de código a escribir, ya que al realizar las pruebas, se evita la posibilidad de escribir código que luego no se vaya a utilizar. Si bien el tiempo de desarrollo de las funcionalidades del sistema se prolonga, estamos invirtiendo tiempo en desarrollar código de calidad y evitando el uso innecesario de herramientas como el depurador.

Empleamos técnicas de desarrollo Clean Code con el objetivo de escribir mejor código. Para nombrar nuestras clases utilizamos nombres significativos vinculados a la realidad que deseamos representar (**Client**, **Figure**, **Material**), y en casos especiales palabras como **Controller**, o **Scanner**, porque son significativas en el contexto en que se emplean. Mantuvimos las funciones pequeñas, intentando que involucren solamente un nivel de abstracción y creando funciones auxiliares con nombres mnemotécnicos con la

intención de que el código se lea como prosa. Siendo que nuestro código es autodescriptivo, desaparece la necesidad de incluir comentarios. Formateamos el código priorizando su legibilidad y evitando que las líneas se extiendan horizontalmente. Ubicamos las constantes arriba, así como la declaración de variables de instancia y los métodos públicos y más relevante, y de igual forma trabajamos los strings, que son definidos como constantes al principio de las clases. Por haber usado la metodología TDD, tuvimos la posibilidad de aplicar refactoreo constante. Nuestro código es idiomático, porque sigue los estándares establecidos por **Microsoft**. Si bien nos vimos tentados a declarar nuestras constantes con mayúsculas separadas por barra baja, decidimos seguir las recomendaciones del lenguaje y usamos **Pascal Case**.

Optimización del motor

➔ *Multihilo*

El orden del motor que se encarga de renderizar es **O(X * Y * S * P)**, correspondiendo a la resolución horizontal, la vertical, los samples per pixel (antialiasing), y las figuras posicionadas.
Esto genera que en **valores grandes**, *ya sea mucha resolución, muchas figuras en la escena, o un valor alto de vectores disparados para difuminar*, el tiempo de renderizado se vuelve **exponencial**.

La solución que se le encontró a esto fue hacer uso de los **múltiples hilos** que tienen los procesadores en la actualidad. Si bien esto no hace que el orden de ejecución se reduzca, a efectos prácticos **reduce drásticamente el tiempo de renderizado**.

Para poder aprovechar todos los hilos disponibles del CPU se hace uso de un **Parallel.For**. Como su nombre indica, es un iterador como el **for tradicional**, con la diferencia de que no se itera de uno en uno, sino que cada hilo se encarga de hacer una iteración mientras los demás hacen las siguientes.

➔ *Problemas - ubicación*

A la hora de aprovechar los **múltiples hilos del procesador**, es necesario saber dónde ubicar la orden. En un comienzo se puso el Parallel.For cuando se guarda el renderizado en memoria, pero esto, por contradictorio que parezca, **volvió el proceso aún más lento** de lo que era. Esto es debido a que cada iteración se realizaba en un tiempo muy reducido, por lo que los hilos debían estar **buscando constantemente la próxima iteración disponible**. El problema radica en que ese tiempo de búsqueda resultaba ser más largo que lo que se demoraba en iterar, por lo que el resultado final era contraproducente.

Una vez entendiendo como funciona, se buscó la posición en la que más se aprovechen los múltiples hilos. Esta ubicación terminó siendo el **primero de los tres *for*** que se recorren para renderizar la imagen. Este asigna cada hilo a una iteración disponible, cada una de estas contiene dentro dos **for tradicionales**, por lo que cada hilo se encarga de recorrer *(resolución de x * samples per pixel)* veces antes de buscar la nueva iteración disponible, siendo el tiempo de búsqueda notoriamente menor al tiempo de iteración.

➔ *Problemas - random*

Al querer llamar a la función tradicional para recibir un número pseudoaleatorio había ocasiones en las cuales **el programa quedaba en loop**.
Luego de un tiempo de investigación descubrimos que los multihilos pueden generar este tipo de problemas **si el programa no está preparado**. Esto se debe a que al pedir un número pseudoaleatorio, se utiliza lo que se conoce como **semilla** *o seed en inglés,* que por defecto está vinculada a la hora del sistema. Con la función tradicional **esta semilla es única una vez que se genera**, causando que los múltiples hilos puedan **generar el mismo número** si el cálculo se realiza con una diferencia de tiempo muy pequeña entre sí.

La solución al problema fue preparar al sistema haciendo uso de una función diferente a la tradicional:

ThreadLocal<Random> random = new ThreadLocal<Random>(() => new Random());

La diferencia de ésta con respecto a la tradicional se basa en que esta le asigna **una *seed* única a cada hilo** evitando completamente la generación de dos número idénticos.

➔ *Creación del archivo final*

Si bien el tiempo de renderizado fue drásticamente reducido gracias al uso de los múltiples hilos, el tiempo de guardado a memoria seguía siendo igual de lento, resultando que no se puedan apreciar las mejoras en el tiempo de renderizado.

A la hora de guardar se *"traduce"* el pixel actual a una variable de tipo string, la cual se añade al texto completo. Esto se puede lograr haciendo uso del asignador (**+=**), manteniendo así el string anterior y agregandole la nueva fila.
El problema es que este asignador lo que hace es reservar un nuevo lugar de memoria para la nueva línea, acceder a este, para finalizar por buscar el lugar de memoria que contiene el string completo y agregarlo.
Estas nuevas reservas de memoria, cada vez que se agrega una línea, hace que buscar lugares para guardar sea cada vez más lento.
Usando el constructor ***StringBuilder*** solucionamos esto.

Justificación de diseño

Diagrama de paquetes

Para la realización, tanto del Diagrama de Paquetes como del Diagrama de Clases, utilizamos diagrams.net que es un software que funciona de manera online y de forma gratuita, facilitandonos así la colaboración entre los integrantes. Debido a la alta resolución de las imágenes del diagrama solicitamos, en caso de ser necesario, hacer zoom para facilitar la lectura.

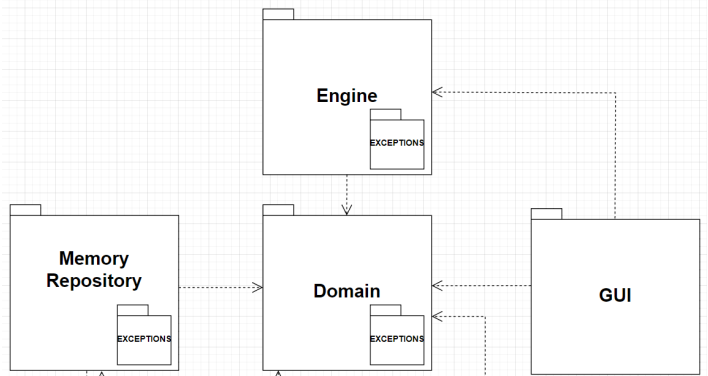
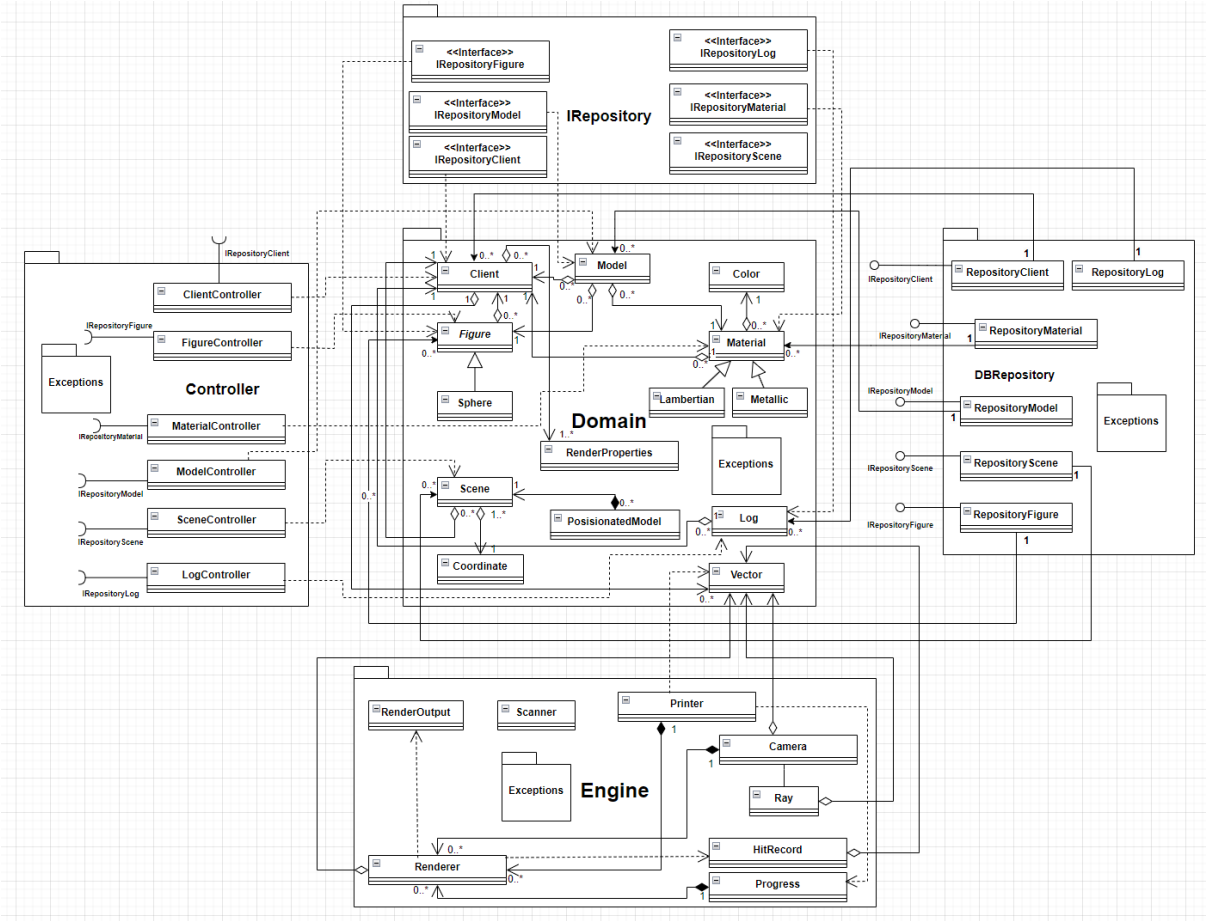
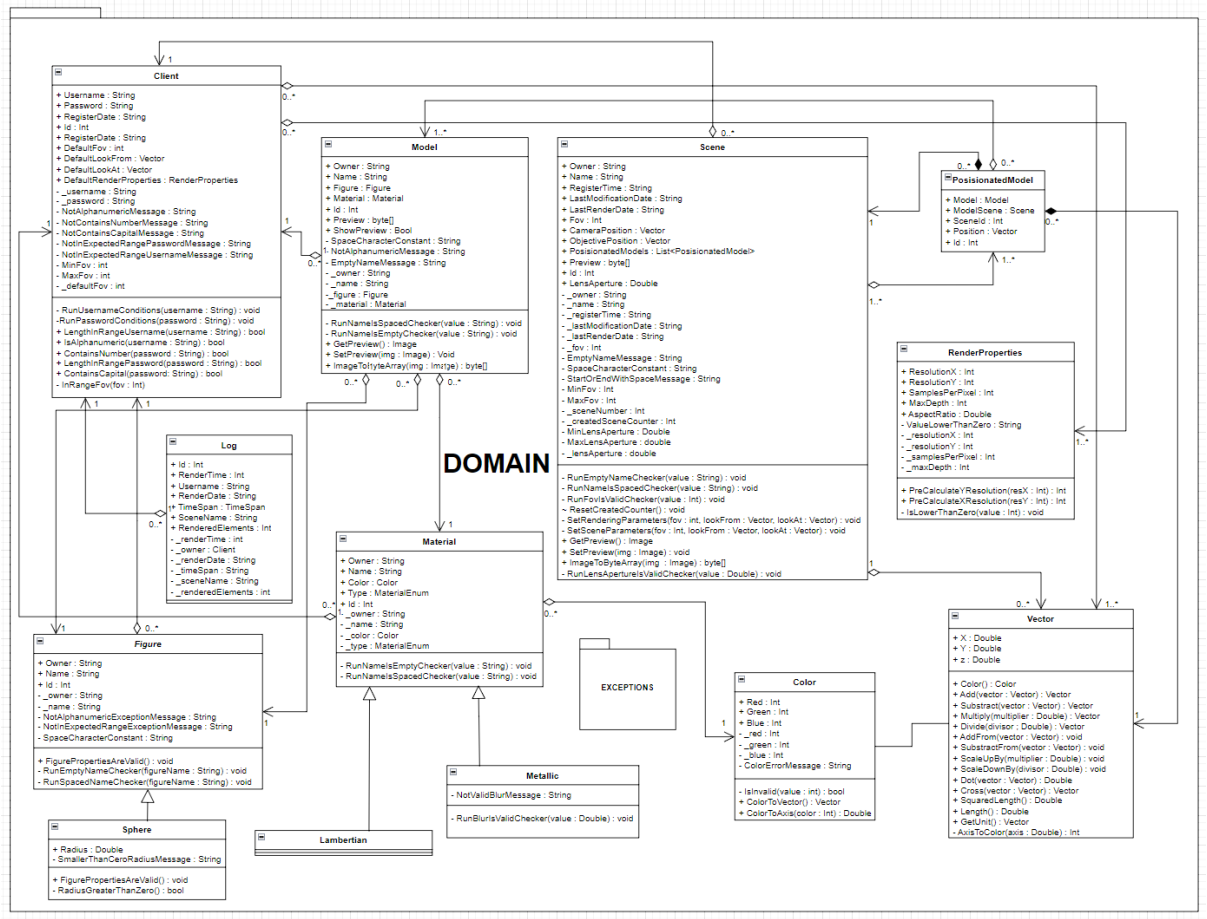


Diagrama de clases

Para facilitar el entendimiento del mismo, realizamos un Diagrama de Clases esquelético que representa la interacción entre las clases de los diferentes paquetes sin sus atributos ni métodos. Posteriormente, dividimos los Diagramas de Clase por paquetes. En el caso de las clases abstractas, sus nombres están identificados por ser escritos con letra cursiva mientras que las interfaces se caracterizan por estar escritos de la siguiente manera: <<interface_name>>.

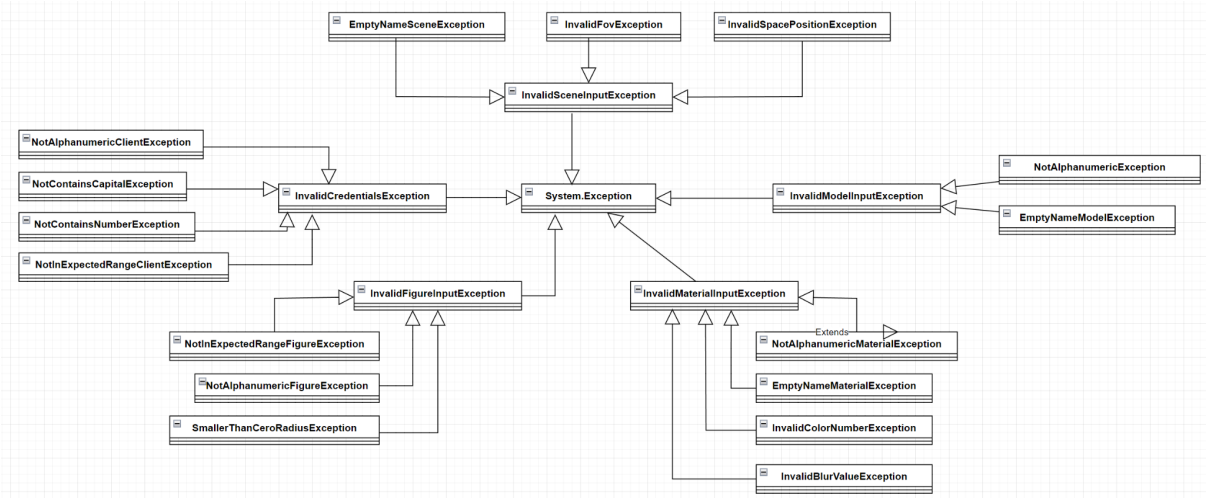


Paquete Domain



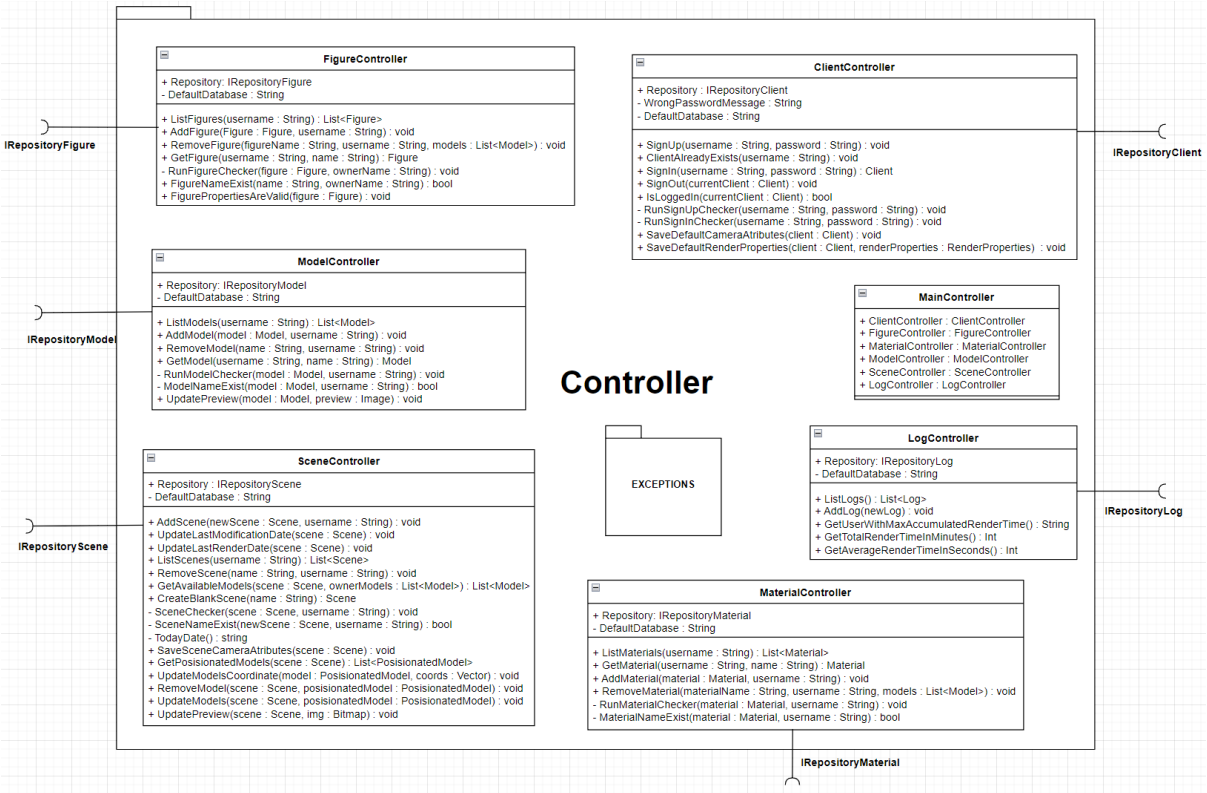
En este paquete se encuentran todos los objetos básicos que utiliza nuestra aplicación con sus respectivas restricciones. Decidimos implementar la clase Figure de forma abstracta y crear una clase Sphere que la extienda, ya que para esta ocasión solo se nos pide representar esferas. En caso de que se quiera agregar otro tipo de figura, solo se debe crear una clase por su nombre y extender la clase Figure, que contiene los atributos básicos que toda figura debe poseer y así evitar repetir código innecesariamente.

Paquete Domain.Exceptions



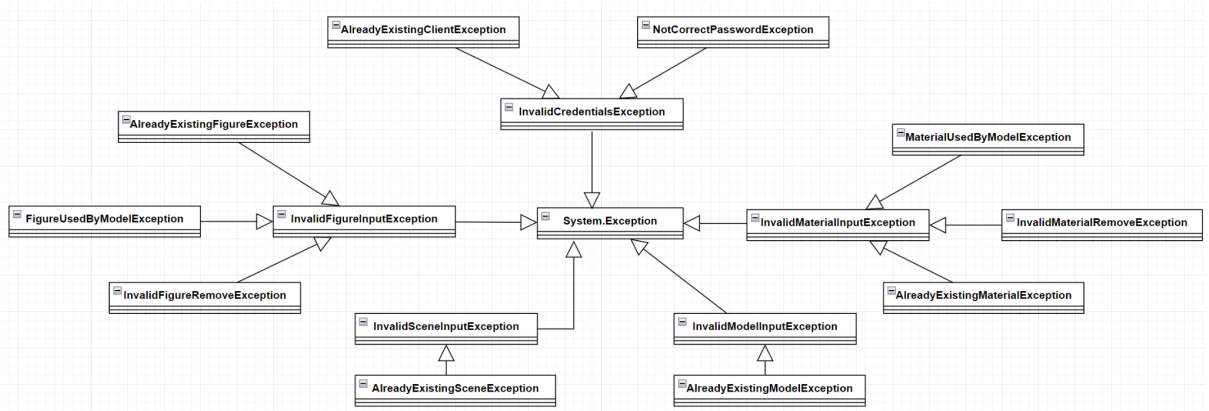
En este paquete se encuentran todas las excepciones necesarias para que cada objeto valide sus atributos. Definimos, por cada modelo, una excepción única que hereda de System.Exception con el fin de catchear nuestras excepciones y no todas las que contiene Exception.
Intentamos utilizar excepciones lo más específicas posibles para así tener mayor control sobre los errores.

Paquete Controller



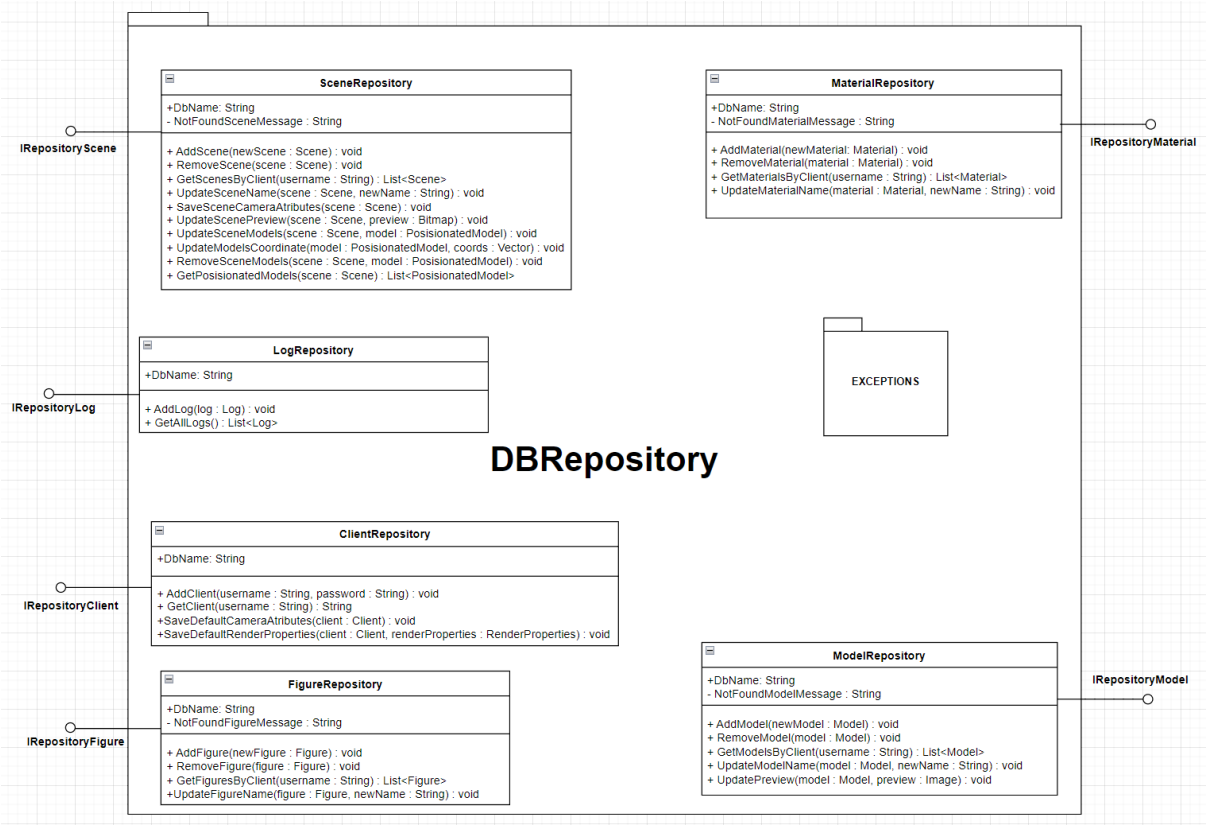
Este paquete maneja la lógica de negocio incluyendo las excepciones ligadas a la misma. Cada “Controller” lleva asociadas excepciones, las cuales heredan de su respectiva excepción padre ubicada en el paquete Domain.Exceptions.

Paquete Controller.Exceptions



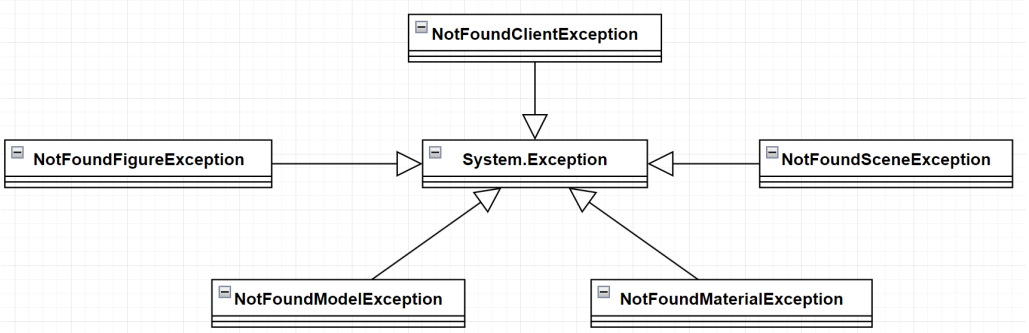
Este paquete nace de la necesidad de poder controlar los posibles errores que pueden ocurrir en la lógica de negocio.

Paquete DBRepository



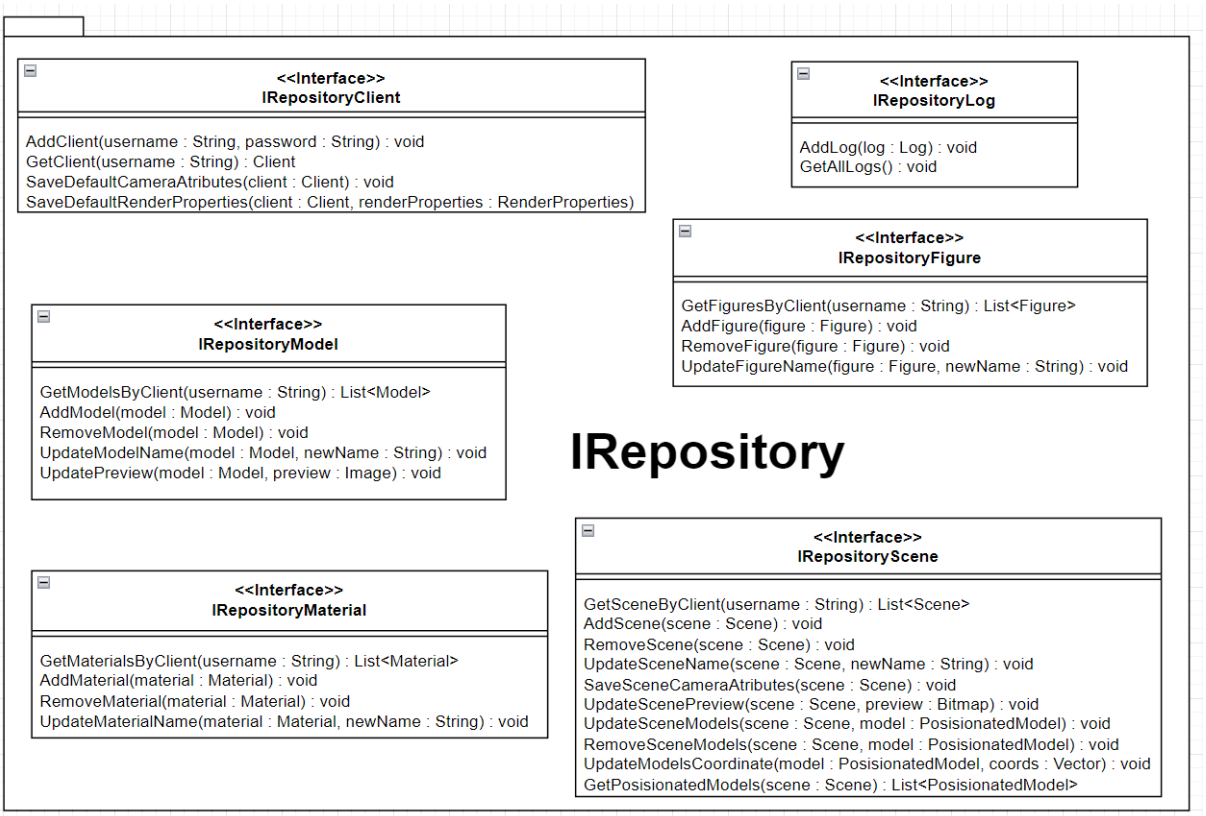
Este paquete se encarga de la implementación en memoria. Por cada repositorio se encuentran los métodos asociados a la agregación, eliminación y obtención de los modelos en el repositorio. Además, implementan su respectiva interfaz.

Paquete DBRepository.Exceptions



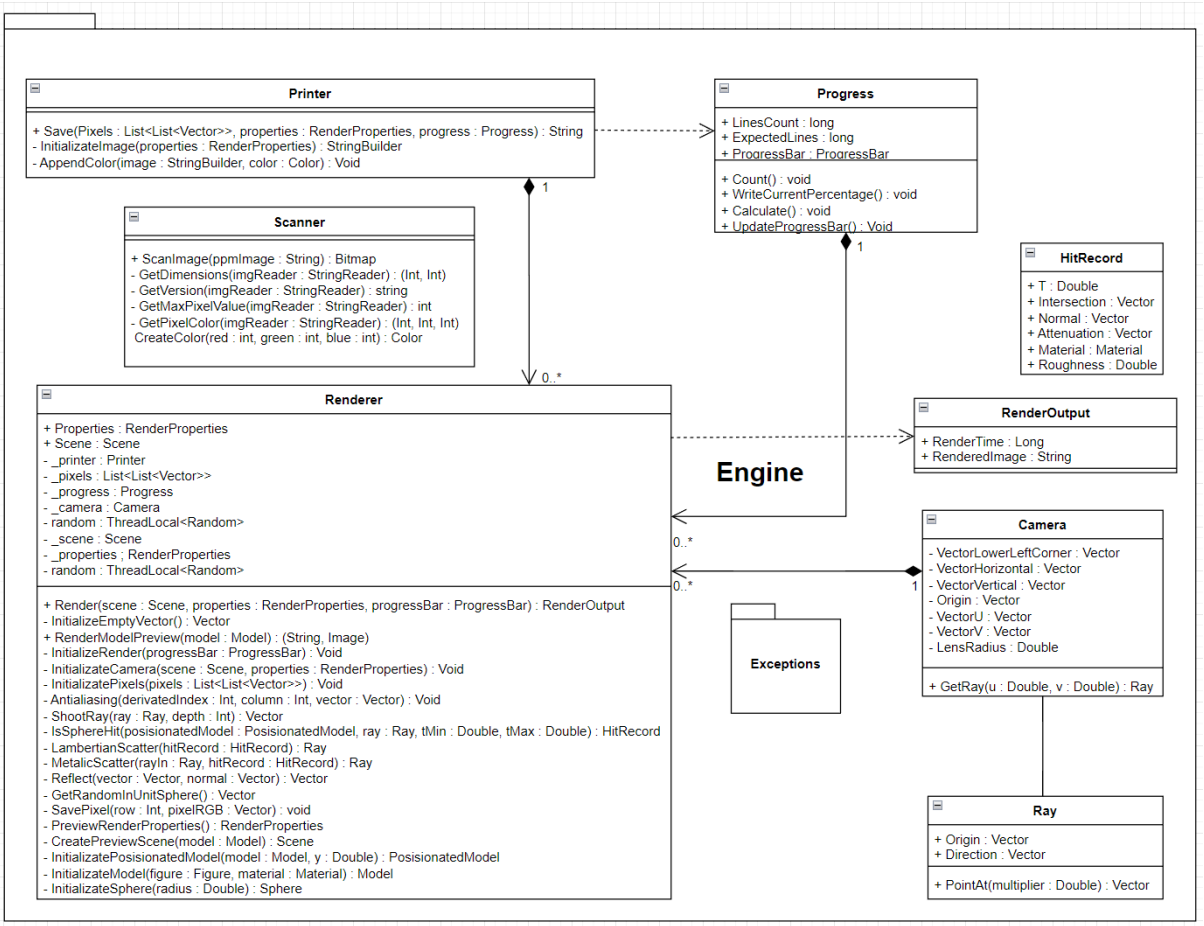
Este paquete incluye las excepciones asociadas a los posibles errores que pueden ocurrir al momento de realizar operaciones en memoria.

Paquete IRepository



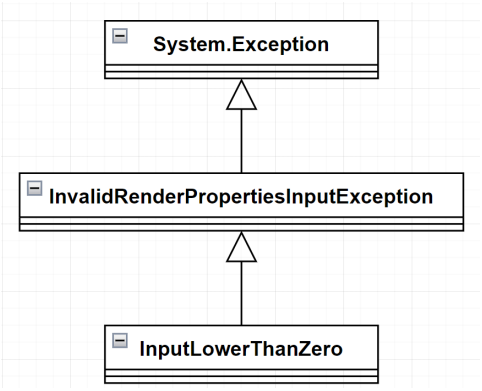
Este paquete contiene las respectivas interfaces por cada modelo y sus métodos.

Paquete Engine



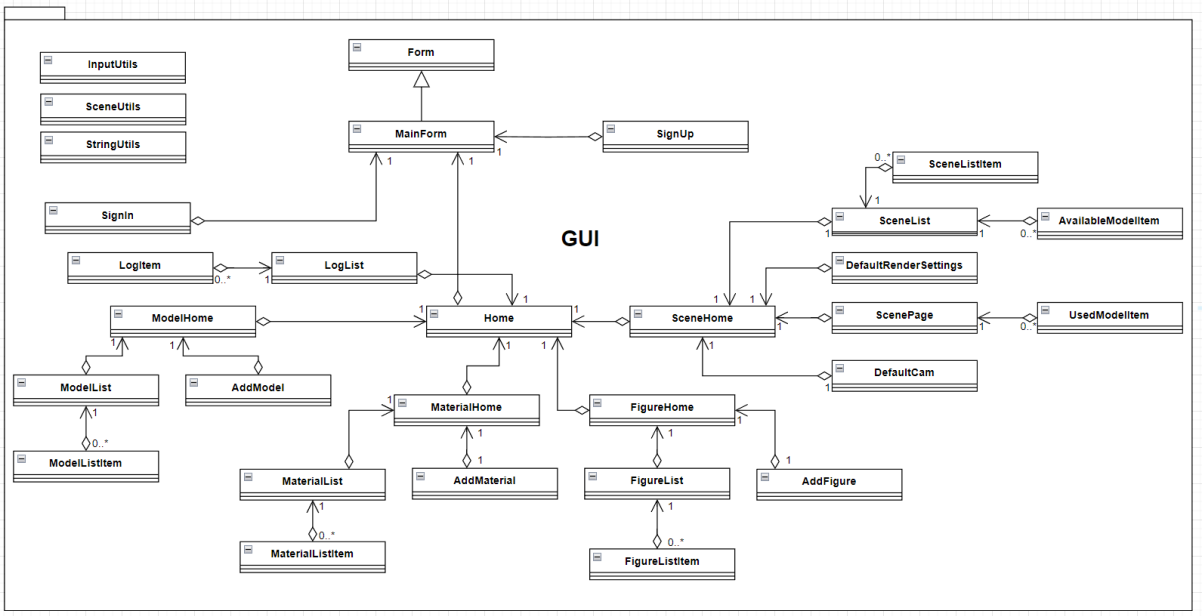
Este paquete contiene todas las clases necesarias para el funcionamiento del motor gráfico y sus relaciones.

Paquete Engine.Exceptions

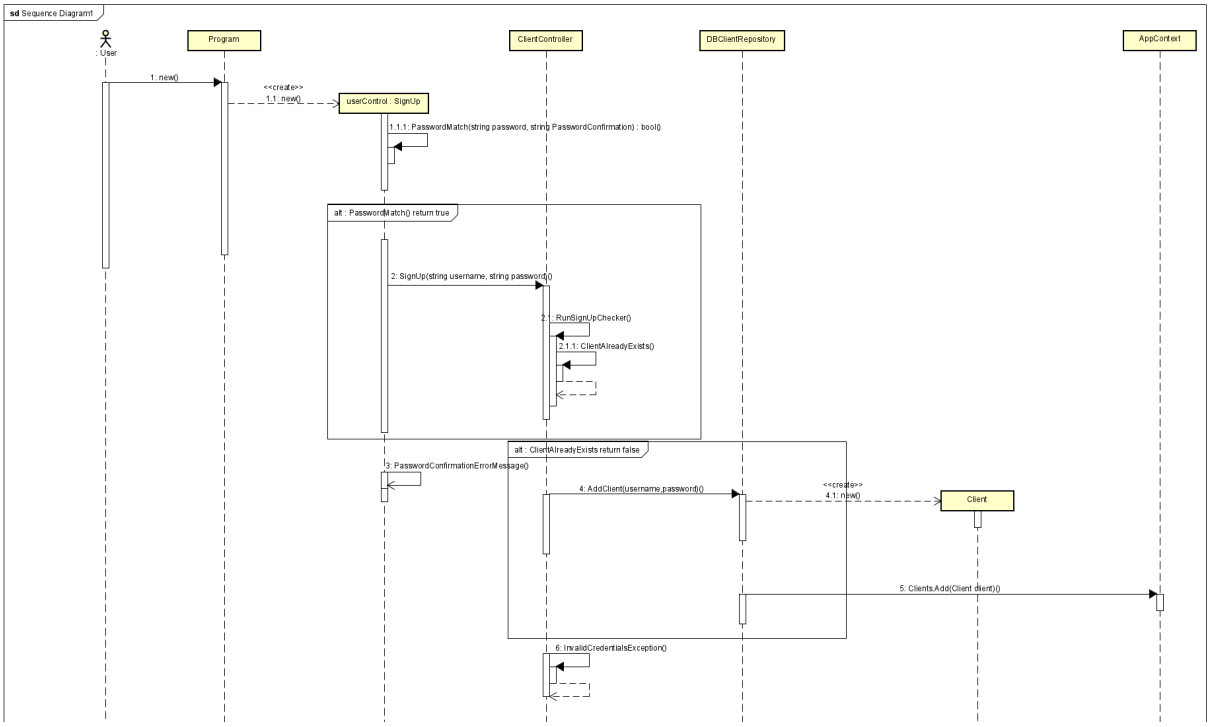


Contiene las excepciones correspondientes al funcionamiento del motor gráfico. En este caso, las únicas restricciones que se aplican son sobre la clase RenderProperties.

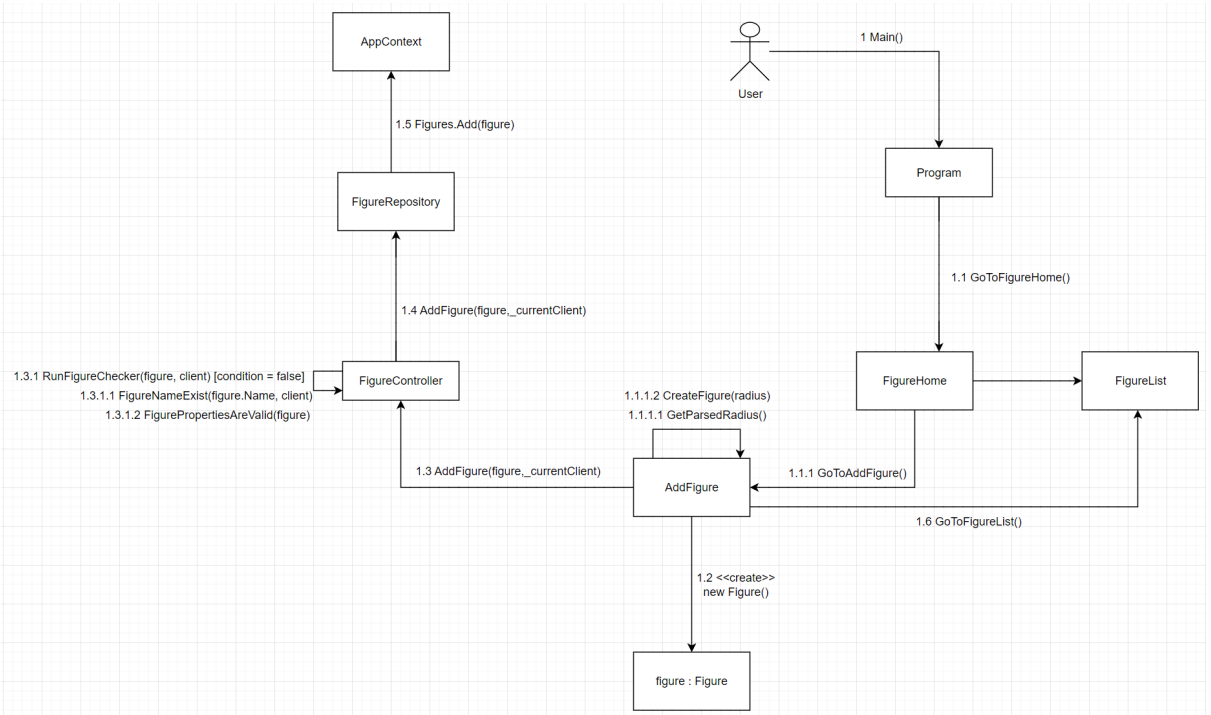
Paquete GUI


GUI
InputUtils
SceneUtils
StringUtils
Form
MainForm
SignUp
SignIn
LogItem
LogList
ModelHome
Home
SceneHome
ModelList
AddModel
ModelListItem
MaterialHome
FigureHome
MaterialList
AddMaterial
FigureList
AddFigure
MaterialListItem
FigureListItem
SceneListItem
SceneList
AvailableModelItem
DefaultRenderSettings
ScenePage
UsedModelItem
DefaultCam

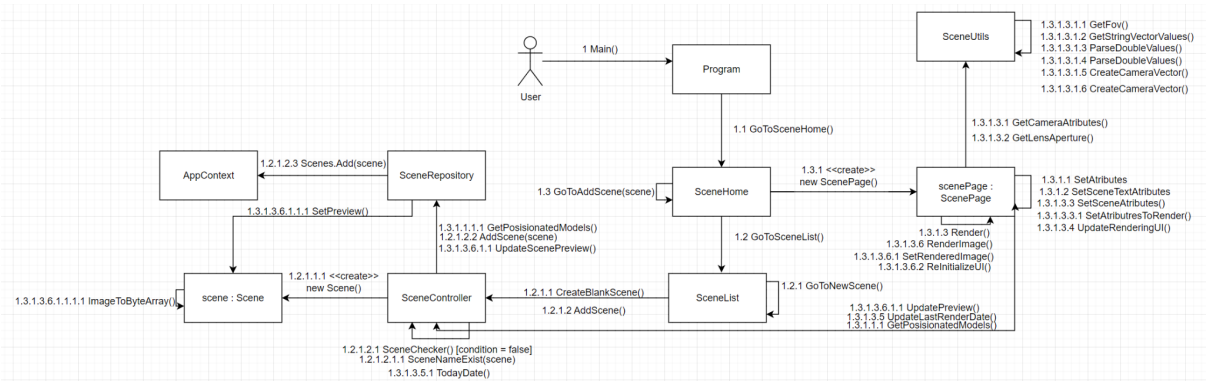
Flujo de SignUp



Flujo de creación de una figura



Flujo de renderizado de una escena



Mecanismos y decisiones de diseño

Basamos nuestro desarrollo en la separación de los modelos, la lógica de negocio y su representación. De esta forma nuestro paquete Models contiene las entidades de nuestra aplicación, que son representadas por el paquete GUI y manejados por el paquete Controller. Este paquete encapsula la lógica de negocio y solicita datos a los modelos y los comunica a la interfaz gráfica. Ni el modelo ni el controlador se preocupan por la representación de los datos, esto es responsabilidad únicamente de la interfaz gráfica. Haciendo empleo de esta arquitectura, desarrollamos software de fácil organización, que resulta escalable y mantenible.

Nuestra vista, desarrollada en Windows Forms, crea instancias de los diferentes controladores a través de los cuales consulta los datos de los modelos. Estos datos son almacenados en bases de datos por clases llamadas repositorios que implementan interfaces a las cuales acceden los controladores. De esta forma, al exponer solamente la interfaz, la implementación es transparente para los controladores y generamos independencia, minimizando el acoplamiento entre las clases y facilitando el potencial intercambio de módulos.

Para el caso de las figuras, desarrollamos una implementación polimórfica que facilite la integración de nuevas figuras en próximas iteraciones del producto. Si bien solo tenemos esferas, introdujimos la clase abstracta **Figure**, con atributos **Owner** y **Name**, así como el método abstracto **FigurePropertiesAreValid**, de tal forma que aquellas figuras que hereden de **Figure**, en este caso la clase **Sphere**, implementan este método con sus validaciones correspondientes. En el caso particular de la esfera, se valida que el radio sea mayor que cero.

Para el manejo de excepciones, creamos excepciones particulares y personalizadas para las diferentes situaciones indeseables que presentan las reglas de negocio. Las agrupamos según los objetos a los que refieren, es decir, se agrupan las excepciones de cliente, de figura, entre otras. Dentro de cada grupo, se genera una excepción padre que hereda de la clase **Exception**, y excepciones hijas que heredan de esta. De esta forma, las excepciones hijas llaman al constructor de la excepción padre, que llama al constructor de la clase **Exception**. Haciendo esto logramos que cuando el programa lanza cualquier excepción hija, podamos atrapar la excepción esperando su padre, y acceder al mensaje original.

Asignación de responsabilidades

Para asignar las responsabilidades en nuestra aplicación, creamos siete paquetes, cuya justificación detallamos a continuación:

Domain

En este paquete definimos los modelos que usa nuestra aplicación, es decir, los objetos básicos que la conforman y sus restricciones. Primeramente consideramos que las validaciones de los objetos corresponden al paquete de lógica, pero esta decisión nos llevó a desarrollar objetos anémicos que carecían de lógica de negocio, por lo que finalmente concluimos que son los mismos objetos los responsables por sus validaciones.

IRepository

Este paquete es el encargado de generar las interfaces que serán implementadas por las clases que almacenan los objetos en nuestra aplicación. Es conveniente generar una interfaz que esconda la implementación de aquellos que usan los repositorios, de forma que si cambiamos la implementación, la aplicación continúa funcionando perfectamente.

DBRepository

En este paquete se implementan los repositorios en bases de datos que implementan sus correspondientes interfaces. Esta es la implementación actual, que anteriormente funcionaba en memoria. Haber usado interfaces, favoreciendo el patrón de variaciones protegidas, nos permitió extraer el antiguo módulo (implementación en memoria) y sustituirlo por uno nuevo que implementa la interfaz persistiendo los datos, sin cambiar el funcionamiento de la aplicación.

Controller

El paquete Controller se encarga de manejar la lógica del negocio, generando las instancias de los repositorios en memoria y manipulando los objetos pertinentes, es decir, desarrollando las inserciones y eliminaciones, y las comprobaciones vinculadas a otras instancias de los mismos tipos.

Engine

Este paquete tiene como finalidad encapsular toda la lógica detrás del funcionamiento del motor gráfico. Involucra las clases que se encargan tanto de configurar las propiedades de renderizado, la conversión de formato PPM a BitMap, y el propio renderizado de la imagen. Decidimos que todas estas funcionalidades corresponden a su propio paquete porque entendemos que nuestra lógica de negocio debe ser ajena al proceso de renderizado, y suponemos que en una aplicación escalable este proceso sería desarrollado por un servicio externo. Asimismo, se encarga de exportar las imágenes en tres formatos distintos. Decidimos implementarlo siguiendo el patrón Strategy, definiendo una estrategia, exportar, y tres estrategias concretas, exportar como jpg, png y ppm. De esta forma contribuimos a tener una solución más polimórfica que respeta el principio OCP de SOLID, pues agregar nuevos formatos para exportar no involucra editar el código ya escrito, sino crear una nueva estrategia concreta.

GUI

El paquete GUI se encarga de contener la interfaz gráfica. Es el punto de inicio de la aplicación. Hace uso de un Form principal, a partir del que crea clases que escuchan los eventos disparados por el usuario, y alertan a los controladores para que actualicen los modelos. Se enfoca en la interacción entre el usuario y la aplicación. Las entidades de nuestro dominio se crean aquí, porque cada formulario contiene la información necesaria para su generación. De esta forma respetamos el patrón creador, pues las entidades son instanciadas por aquella clase que tiene la información necesaria para crearlas.

Test

Basados en aplicar las técnicas de TDD, el paquete Test contiene los diferentes tests unitarios que implementamos mientras desarrollamos la aplicación. Estas pruebas nos aseguran el correcto funcionamiento del código y a su vez el funcionamiento de las excepciones que rigen al sistema.

Cobertura de pruebas unitarias

Se logró alcanzar una cobertura de código del 91%, esto se debe al intento de seguir estrictamente la metodología de Test Driven Development la cual nos impone no crear código de producción hasta haber creado las pruebas necesarias que lo respalden.
Gracias a la implementación de esta metodología, pudimos percatarnos de varios de sus beneficios durante el proceso de desarrollo, siendo la gran cobertura de código uno de ellos.
Se crearon 302 tests de los cuales todos funcionan sin errores (se adjunta prueba de esto en el anexo).
A continuación se detalla la cobertura por paquete, mostrando la cantidad de bloques y líneas cubiertas por cada uno de los mismos.
Consideramos que los porcentajes obtenidos son más que satisfactorios.

Hierarchy	Covered (Lines)	Covered (%Lines)	Not Covered (Lines)	Not Covered (%Lines)
Renzo_PC-RENZO 2023-06-14 20_05_54.coverage	1694	91,67%	153	8,28%
dbrepository.dll	227	73,46%	82	26,54%
engine.dll	479	93,92%	31	6,08%
controller.dll	433	93,72%	29	6,28%
domain.dll	555	97,88%	11	1,94%

Resultado de la ejecución de pruebas

En el anexo se encuentran los resultados de algunas de las 302 pruebas realizadas.

Benchmarking

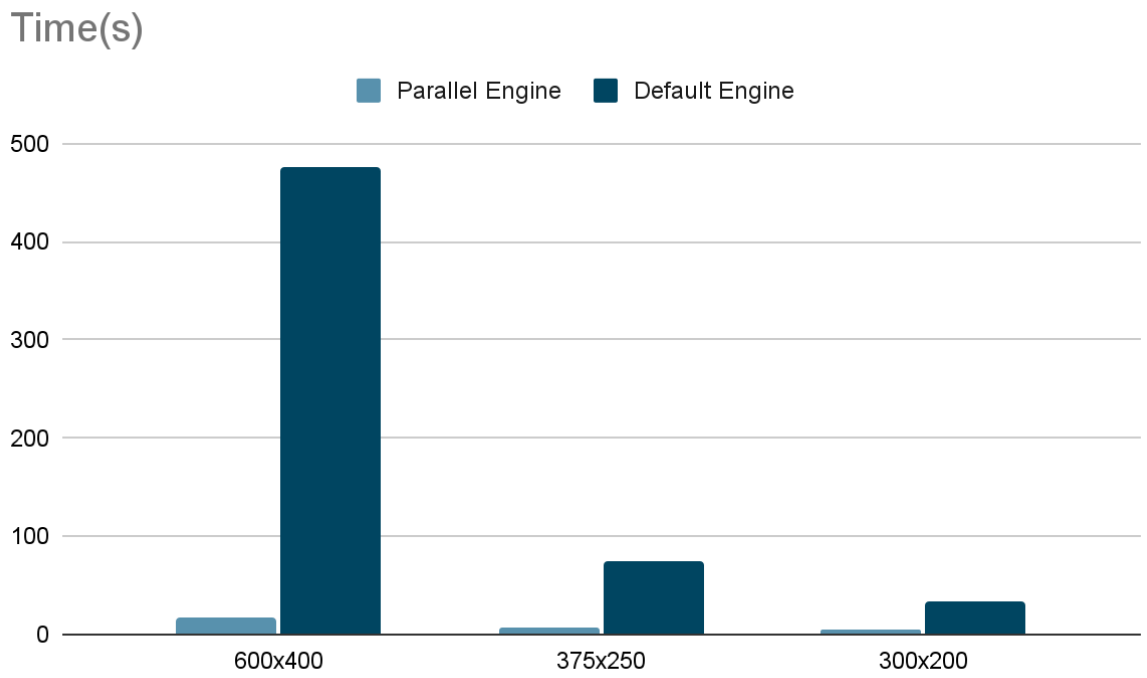
Calculamos la diferencia entre el motor sin uso del Parallel.For y de StringBuilder. Estos fueron los resultados según la resolución recibida.

➔ **Parallel Engine**

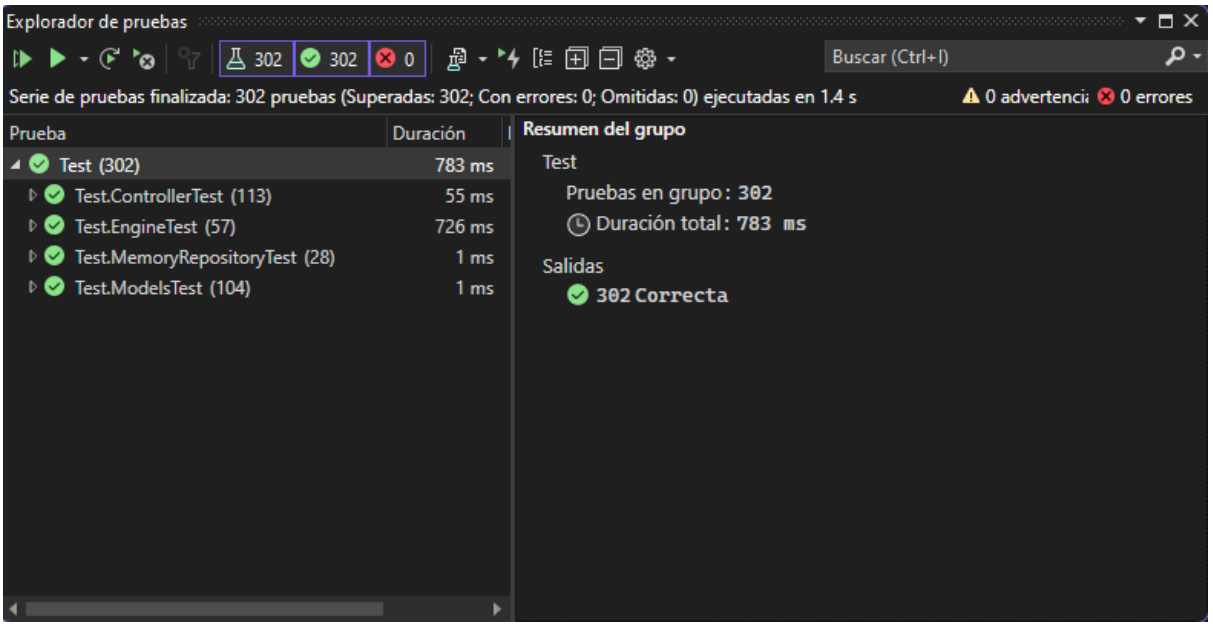
Resolution	Time(s)
600x400	18.1
375x250	7.5
300x200	5.5

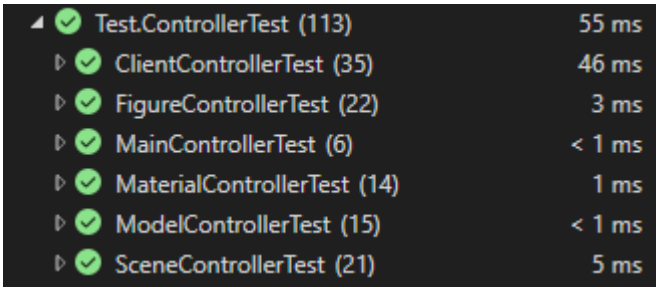
➔ **Standard Engine**

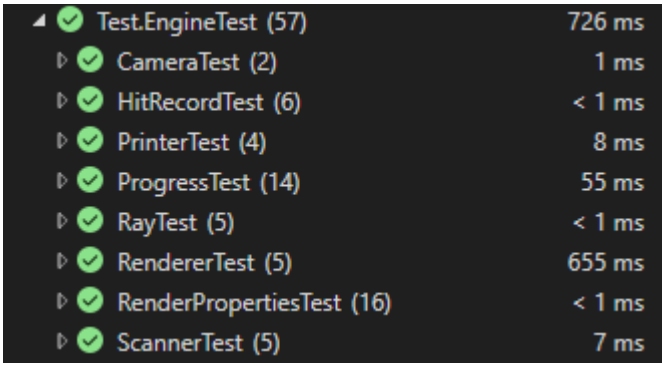
Resolution	Time(s)
600x400	457.6
375x250	75.4
300x200	34.1

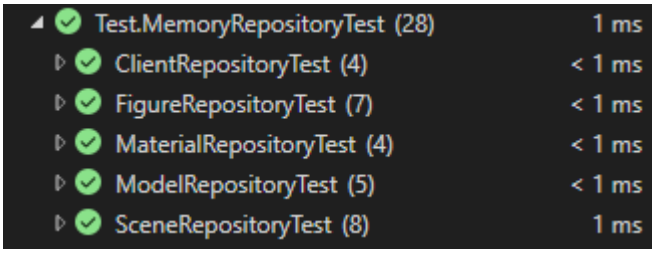


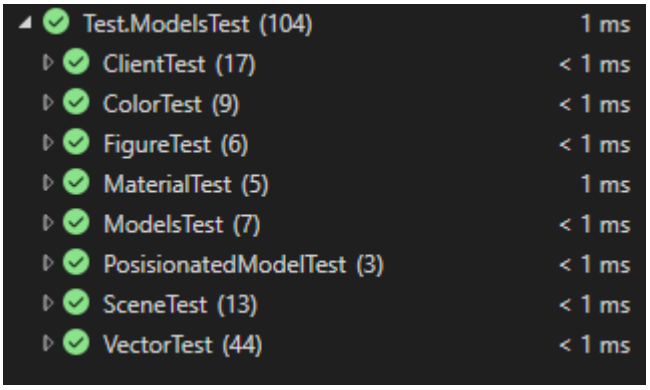
Anexo


Explorador de pruebas
302
302
0
Buscar (Ctrl+I)
Serie de pruebas finalizada: 302 pruebas (Superadas: 302; Con errores: 0; Omitidas: 0) ejecutadas en 1.4 s
0 advertenci
0 errores
Prueba
Duración
Test (302) 783 ms
Test.ControllerTest (113) 55 ms
Test.EngineTest (57) 726 ms
Test.MemoryRepositoryTest (28) 1 ms
Test.ModelsTest (104) 1 ms
Resumen del grupo
Test
Pruebas en grupo: 302
Duración total: 783 ms
Salidas
302 Correcta


Test.ControllerTest (113) 55 ms
ClientControllerTest (35) 46 ms
FigureControllerTest (22) 3 ms
MainControllerTest (6) < 1 ms
MaterialControllerTest (14) 1 ms
ModelControllerTest (15) < 1 ms
SceneControllerTest (21) 5 ms


Test.EngineTest (57) 726 ms
CameraTest (2) 1 ms
HitRecordTest (6) < 1 ms
PrinterTest (4) 8 ms
ProgressTest (14) 55 ms
RayTest (5) < 1 ms
RendererTest (5) 655 ms
RenderPropertiesTest (16) < 1 ms
ScannerTest (5) 7 ms


Test.MemoryRepositoryTest (28) 1 ms
ClientRepositoryTest (4) < 1 ms
FigureRepositoryTest (7) < 1 ms
MaterialRepositoryTest (4) < 1 ms
ModelRepositoryTest (5) < 1 ms
SceneRepositoryTest (8) 1 ms




```
namespace Test.MemoryRepositoryTest
{
    [TestClass]
    [ExcludeFromCodeCoverage]

    0 referencias | - cambios | -autores, - cambios
    public class ClientRepositoryTest
    {
        private ClientRepository _clientRepository;

        [TestInitialize]
        0 referencias | - cambios | -autores, - cambios
        public void TestInitialize()
        {
            _clientRepository = new ClientRepository();
        }

        [TestMethod]
        | 0 referencias | - cambios | -autores, - cambios
        public void CreateClientRepository_OkTest()
        {
            _clientRepository = new ClientRepository();
        }

        [TestMethod]
        | 0 referencias | - cambios | -autores, - cambios
        public void AddClientToClientRepository_OkTest()
        {
            _clientRepository.AddClient("Gomez", "GomezSecret123");
        }

        [TestMethod]
        | 0 referencias | - cambios | -autores, - cambios
        public void GetClient_OkTest()
        {
            _clientRepository.AddClient("user", "Password123");
            Assert.AreEqual("user", _clientRepository.GetClient("user").Username);
            Assert.AreEqual("Password123", _clientRepository.GetClient("user").Password);
        }

        [TestMethod]
        [ExpectedException(typeof(NotFoundClientException))]
        | 0 referencias | - cambios | -autores, - cambios
        public void GetClient_NotAddedClient_FailTest()
        {
            Assert.AreEqual("user", _clientRepository.GetClient("user").Username);
        }
    }
}
```

```
[TestClass]
[ExcludeFromCodeCoverage]
0 referencias | Santiago, Hace 3 días | 3 autores, 28 cambios
public class FigureRepositoryTest
{
    private FigureRepository _figureRepository;

    [TestInitialize]
    0 referencias | Renzo-Test, Hace 4 días | 1 autor, 2 cambios
    public void TestInitialize()
    {
        _figureRepository = new FigureRepository();
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 3 cambios
    public void CreateFigureRepository_OkTest()
    {
        _figureRepository = new FigureRepository();
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 6 cambios
    public void GetFiguresByClient_OwnerName_OkTest()
    {
        Figure newFigure = new Sphere()
        {
            Name = "Test",
            Owner = "OwnerName"
        };
        _figureRepository.AddFigure(newFigure);

        Assert.AreEqual(newFigure, _figureRepository.GetFiguresByClient("OwnerName")[0]);
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 3 cambios
    public void GetFiguresByClient_TwoClients_OkTest()
    {
        Figure firstFigure = new Sphere()
        {
            Name = "FigureOne",
            Owner = "OwnerOne"
        };
        _figureRepository.AddFigure(firstFigure);

        Figure secondFigure = new Sphere()
        {
            Name = "FigureTwo",
            Owner = "OwnerTwo"
        };
        _figureRepository.AddFigure(secondFigure);

        Assert.AreEqual("FigureOne", _figureRepository.GetFiguresByClient("OwnerOne")[0].Name);
        Assert.AreEqual("FigureTwo", _figureRepository.GetFiguresByClient("OwnerTwo")[0].Name);
    }
```

```
[TestMethod]
| 0 referencias | Renzo-Test, Hace 4 días | 2 autores, 3 cambios
public void GetFiguresByClient_NotExisting2Client()
{
    _figureRepository.GetFiguresByClient("");
}

[TestMethod]
| 0 referencias | Renzo-Test, Hace 4 días | 2 autores, 9 cambios
public void AddFigure_OkTest()
{
    Figure newFigure = new Sphere()
    {
        Name = "Test",
        Owner = "OwnerName"
    };

    _figureRepository.AddFigure(newFigure);

    List<Figure> iterable = _figureRepository.GetFiguresByClient("OwnerName");

    CollectionAssert.Contains(iterable, newFigure);

}

[TestMethod]
| 0 referencias | Renzo-Test, Hace 4 días | 2 autores, 10 cambios
public void RemoveFigure_OkTest()
{
    Figure newFigure = new Sphere()
    {
        Name = "Test",
        Owner = "OwnerName"
    };

    _figureRepository.AddFigure(newFigure);
    _figureRepository.RemoveFigure(newFigure);
    List<Figure> figures = _figureRepository.GetFiguresByClient("OwnerName");

    Assert.IsFalse(figures.Any());
}
```

```
[TestClass]
[ExcludeFromCodeCoverage]
0 referencias | - cambios | -autores, - cambios
public class ClientTest
{
    private Client _client;

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void CanCreateClient_OkTest()
    {
        _client = new Client();
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void SetUsername_Gomez_OkTest()
    {
        _client = new Client()
        {
            Username = "Gomez",
        };
        Assert.AreEqual("Gomez", _client.Username);
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void SetPassword_GomezSecret123_OkTest()
    {
        _client = new Client()
        {
            Password = "GomezSecret123",
        };
        Assert.AreEqual("GomezSecret123", _client.Password);
    }


    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void SetDefaultFov_OkTest()
    {

        _client = new Client()
        {
            DefaultFov = 55
        };

        Assert.AreEqual(55, _client.DefaultFov);
    }
```

```
[TestClass]
[ExcludeFromCodeCoverage]
0 referencias | - cambios | -autores, - cambios
public class FigureTest
{
    private Figure _figure;
    private Sphere _sphere;

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void CanCreateFigure_OkTest()
    {
        _figure = new Sphere();
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void SetName_Dragon_Balloon_OkTest()
    {
        _figure = new Sphere()
        {
            Name = "DragonBalloon",
        };
        Assert.AreEqual("DragonBalloon", _figure.Name);
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void CanCreateSphere_OkTest()
    {
        _sphere = new Sphere();
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void SetRadius_351_OkTest()
    {
        _sphere = new Sphere()
        {
            Radius = 3.51,
        };
        Assert.AreEqual(3.51, _sphere.Radius);
    }

    [TestMethod]
    | 0 referencias | - cambios | -autores, - cambios
    public void CanCreateSphere_Balloon_351_OkTest()
    {
        _sphere = new Sphere()
        {
            Name = "Balloon",
            Radius = 3.51
        };
        Assert.AreEqual(3.51, _sphere.Radius);
        Assert.AreEqual("Balloon", _sphere.Name);
    }
```

```
[TestClass]
[ExcludeFromCodeCoverage]

0 referencias | Renzo-Test, Hace 4 días | 1 autor, 8 cambios
public class MaterialTest
{
    private Material _material;

    [TestMethod]

    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 6 cambios
    public void CanCreateMaterial_OkTest()
    {
        _material = new Material();
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 4 cambios
    public void SetOwner_Gomez_OkTest()
    {
        _material = new Material()
        {
            Owner = "Gomez",
        };
        Assert.AreEqual("Gomez", _material.Owner);
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 4 cambios
    public void SetName_Brick_OkTest()
    {
        _material = new Material()
        {
            Name = "Brick",
        };
        Assert.AreEqual("Brick", _material.Name);
    }

    [TestMethod]
    | 0 referencias | Renzo-Test, Hace 4 días | 1 autor, 5 cambios
    public void SetColor_validColor_OkTest()
    {
        Color _newColor = new Color();

        _material = new Material()
        {
            Color = _newColor,
        };

        Assert.AreEqual(_newColor, _material.Color);
    }
```