GUIA TALLER #5

Ximena Contreras

Laura Avilés

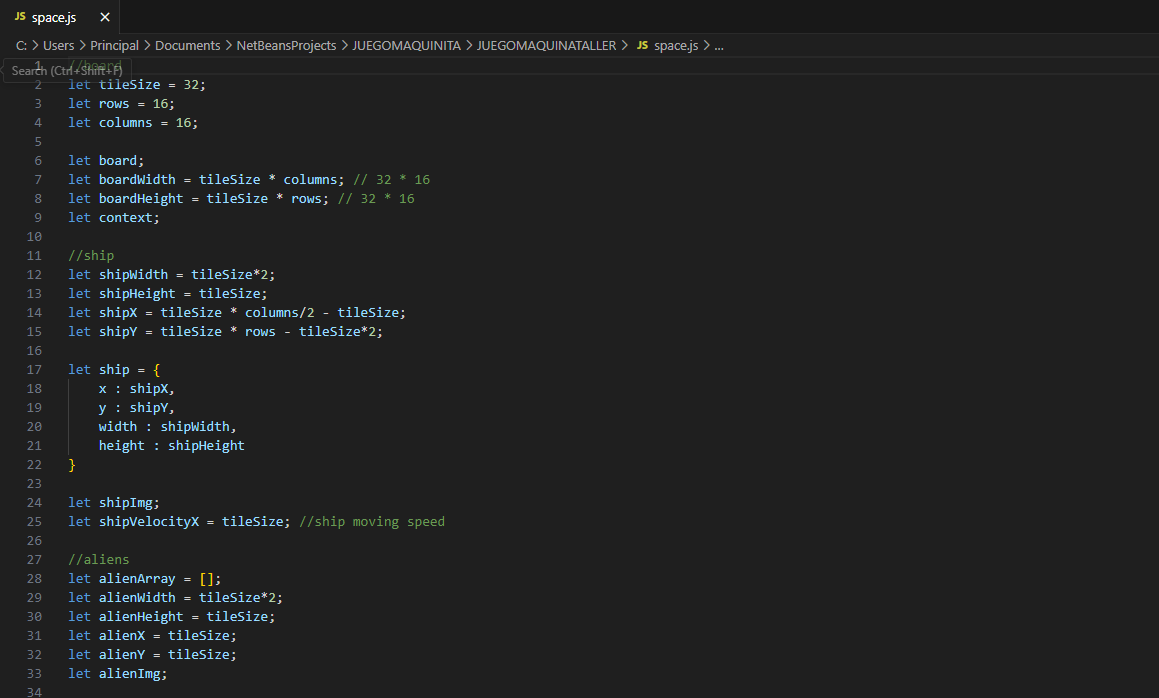
Nicolas Núñez

Taller de Programación

Olga Lucia Roa

Introducción

Codigo



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

ACTIVIDAD DE TRABAJO AUTONOMO

1. Enuncie las características de debe cumplir todo buen componente reusable.

\* Modularidad: el componente debería ser autónomo e independiente; debería haber una división clara entre la funcionalidad interna de un componente y los demás componentes del sistema. Esta característica facilita el proceso de integración de un componente en varios sistemas.

\* Reutilizabilidad: el componente debería permitirse en varios contextos y entornos de la aplicación. Debería ofrecer una funcionalidad lo suficientemente genérica como para ser útil en una variedad de situaciones.

\* Alta cohesión: un componente debería tener responsabilidad única y bien definida o un conjunto de responsabilidades relacionadas entre sí. Esta característica garantiza que el componente sea fácil de entender, mantener y reutilizar.

\* Bajo acoplamiento: un componente no debería tener dependencias de otros componentes o sistemas. Con esta característica, cambia un componente sin ninguna repercusión en otras partes del sistema.

\* Encapsulación: los detalles internos del componente no deben estar expuestos a los componentes externos. Dicho de otra manera, las capacidades del componente solo deben estar habilitadas a través de interfaces bien definidas. Los usuarios pueden interactuar con el componente sin conocer y comprender su implementación interna.

\* Escalabilidad: el componente debe ser capaz de manejar cargas de trabajo en crecimiento o ser modificado para cumplir con diferentes requisitos de rendimiento, independientemente del diseño subyacente.

\* Extensibilidad: El componente, si es necesario, debe poder extenderse o modificarse para cumplir con nuevos requisitos sin requerir la reescritura de todo el sistema.

\* Eficiente: el componente debería hacer su trabajo lo mejor posible y consumir la menor cantidad de recursos. Dado que los componentes reutilizables se reutilizarán, es probable que abusen de la reutilización. Además, tendrán que trabajar con sistemas de alta frecuencia la mayoría de las veces; no está bien que un componente suene cada vez que un sistema al que vivimos intenta interactuar con él.

\*Documentación: un componente reutilizable debería contener una documentación completa de lo que hace, las interfaces disponibles, limitaciones y suposiciones donde sea el caso. Advirtiendo a otros desarrolladores de qué esperar si están integrando para que puedan envolverlo en su sistema.

\*Testabilidad: el componente debe poder probarse en su comportamiento, entradas y salidas solas o en un sistema. entradas / salidas / comportamientos claros y fáciles de probar para verificar que el componente hace lo que se espera.

**Definición de Patrón de Software:**

Un patrón de software es una solución reutilizable a un problema que ocurre comúnmente en el diseño de software. Estos patrones proporcionan un modelo general para resolver desafíos específicos de diseño y pueden adaptarse a varios contextos. Ayudan a los desarrolladores a evitar "reinventar la rueda" y a seguir mejores prácticas para escribir código mantenible, eficiente y escalable.

**Alta Cohesión y Bajo Acoplamiento:**

Alta Cohesión: Se refiere a cuán estrechamente relacionadas y enfocadas están las responsabilidades de un único módulo o clase. Una clase con alta cohesión tiene un propósito claro y específico, lo que la hace más fácil de mantener, probar y entender.

Bajo Acoplamiento: El acoplamiento se refiere al grado de dependencia entre diferentes módulos o clases. Un bajo acoplamiento significa que los módulos son independientes entre sí, lo que hace que el sistema sea más flexible, mantenible y más fácil de modificar sin afectar a otros componentes. En conjunto, la alta cohesión y el bajo acoplamiento crean sistemas que son más fáciles de gestionar y escalar.

**Cinco Patrones de Concurrencia:**

Thread Pool: Un conjunto de hilos precreados que pueden reutilizarse para tareas, reduciendo el costo de creación y destrucción de hilos. Este patrón es ideal para aplicaciones donde hay muchas tareas cortas que deben manejarse de manera concurrente.

Productor-Consumidor: Involucra dos entidades: un productor que crea datos o tareas y un consumidor que los procesa. Este patrón es útil para gestionar cargas de trabajo donde la producción y el consumo de recursos operan a diferentes velocidades.

Futuro (o Promesa): Este patrón permite la ejecución de una tarea en segundo plano y devuelve una "promesa" del resultado, que puede resolverse más adelante. Ayuda a evitar bloquear el hilo principal mientras se espera a que una tarea se complete.

Monitor de Objeto: Este patrón asegura que los métodos dentro de un objeto puedan ser accedidos de manera segura por múltiples hilos. Involucra acceso sincronizado a recursos compartidos para prevenir conflictos e inconsistencias.

Balking: En este patrón, un método devuelve sin ejecutar si determina que el estado actual del objeto no es adecuado para la acción. Es útil en situaciones donde no se desea bloquear un hilo, pero la operación no puede proceder.

**Thread Pool como un Patrón de Concurrencia:**

El patrón de Thread Pool se considera un patrón de concurrencia porque aborda el problema de gestionar y reutilizar eficientemente múltiples hilos en un entorno concurrente. En lugar de crear y destruir hilos para cada tarea, un pool de hilos mantiene un conjunto de hilos reutilizables que pueden manejar múltiples tareas con el tiempo. Esto minimiza la sobrecarga asociada con la gestión de hilos, asegura una mejor asignación de recursos y mejora el rendimiento en sistemas concurrentes donde deben ejecutarse múltiples tareas simultáneamente.

PREGUNTAS ORIENTADORAS

¿Cuáles fueron los aprendizajes obtenidos al realizar esta guía?,

Obtuvimos un aprendizaje la hora de desarrollar esta guía gracias a el desarrollo de un juego como el de spacetrigger, afianzamos nuestros aprendizajes con el desarrollo del trabajo autónomo y la inducción de la guía.  
¿Dónde presento mayor dificultad resolviendo la guía?

¿cómo lo resolvieron? ¿cuáles fueron las estrategias de solución?

Como idea principal no teníamos un juego en mente, así que buscamos diferentes alternativas que nos ayudaran a crear un juego funcional y que se adaptara a los requerimientos planteados.

Para la creación usamos varias investigaciones y bases de conocimientos pasados para obtener este resultado final.