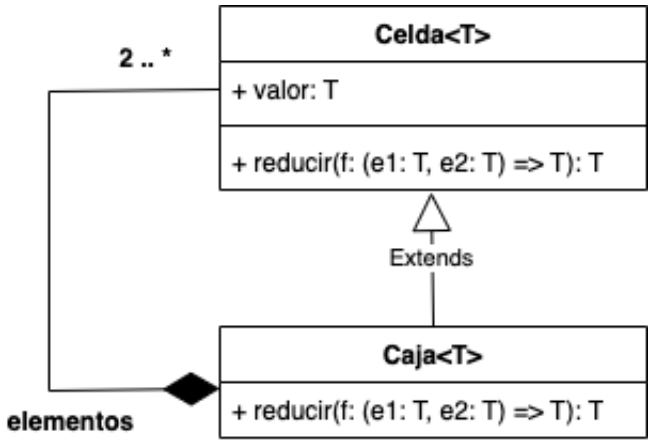
Desarrollo de Software
Prof. Carlos Alonzo
Primer Examen Parcial
09-05-2023 **Nombre:** _____

El examen es individual. Lea con detenimiento y tranquilidad cada uno de los enunciados. Justifique sus respuestas de manera clara y resumida.

1 Pregunta 1: Código genérico (5 puntos)

En el Desarrollo de Software, una de las cualidades deseadas (y también un requerimiento no funcional) es la ***Reusabilidad***. Existen diversas técnicas para alcanzar la reusabilidad de código o de un diseño, una de ellas es la ***Genericidad*** o ***Programación Genérica***.

Lea con detenimiento el siguiente fragmento de diseño que se le presenta en un **Diagrama de Clases UML**.



El dominio de la programación genérica es fundamental, ya que hoy en día está difundida tanto en el desarrollo de software de *backend* como de aplicaciones móviles o web.
Por tal motivo, usted debe demostrar que domina la genericidad. Para ello, usted debe implementar en **TypeScript/Dart** el método ***reducir*** de las clases genéricas **Celda** y **Caja**. Este método se encarga de aplicar una función de reducción cualquiera en el dominio de los valores de tipo **T**.

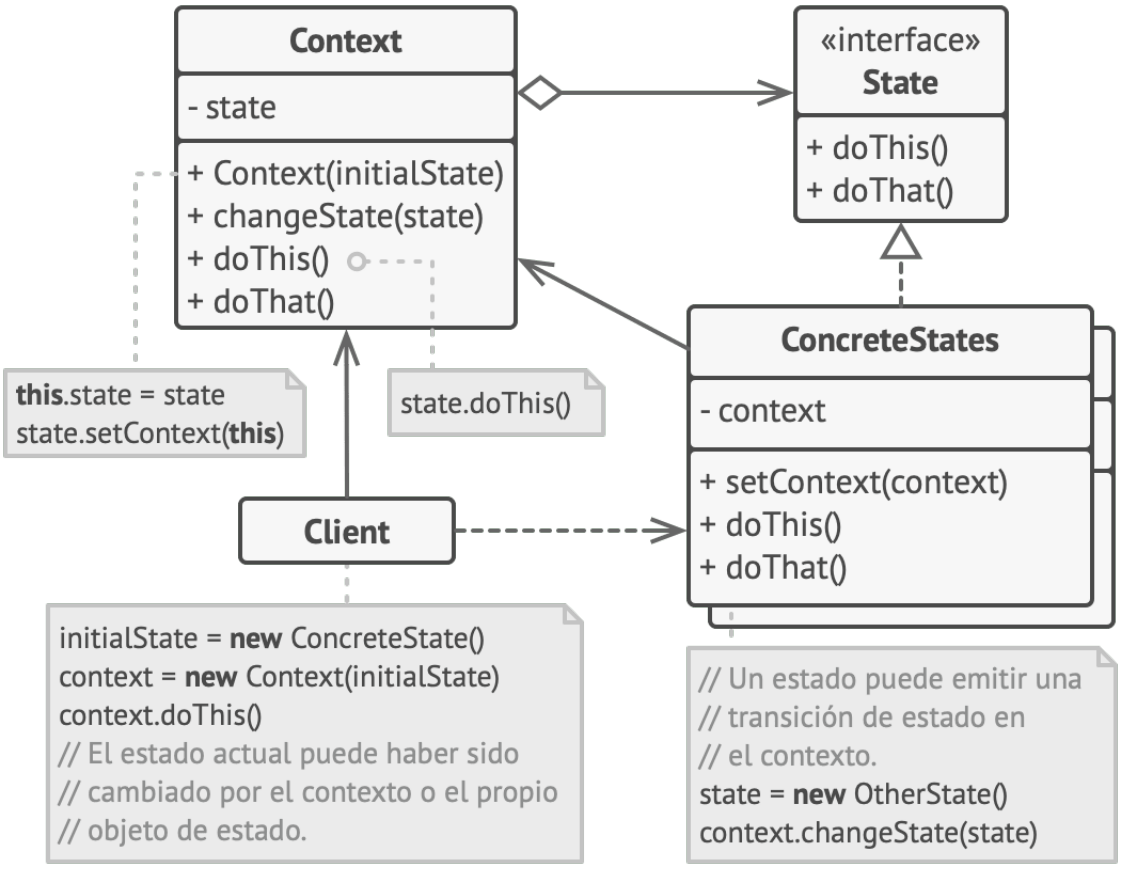
Por ejemplo, suponga que el tipo **T** es un **number**, entonces una función de reducción (de las millones que pueden existir) podría ser: la suma de dos números que genera un nuevo número. De esta manera, si se le pasa la función suma al método **reducir**, entonces se encargaría de sumar todos los valores numéricos que se encuentren en las cajas y celdas.

Observe con atención que en el diagrama de clases se evidencia la existencia de **Polimorfismo de Subtipos** y **Polimorfismo Paramétrico**, y además se usa el **Patrón de Diseño Composición (Composite)**.

Desarrollo de Software
Primer Examen Parcial

2 Pregunta 2: SOLID y Patrones de Diseño (4 puntos)

Explique con un ejemplo claro y concreto, por qué el patrón de diseño **Estado** favorece el cumplimiento del ***Principio Abierto-Cerrado (OCP)*** de **SOLID**. También debe explicar por qué este patrón hace uso de la **composición con delegación**.



3 Pregunta 3: A Gift ! (4 puntos)

A continuación se describe un problema de la vida real. Lea con detenimiento lo que se plantea. Usted debe modelar su diseño elaborando un **Diagrama de Clases UML** y debe justificar con claridad el **Patrón de Diseño** que empleará para dar solución al problema planteado:

1. Se desea implementar un sistema de archivos simple en forma de árbol (similar a Windows o LINUX) para un sistema operativo

2. Usted está tomando varias decisiones de diseño

3. Decisión 1: Todo el sistema de archivos está representado por un objeto de sistema de archivos que contiene una referencia a la carpeta padre

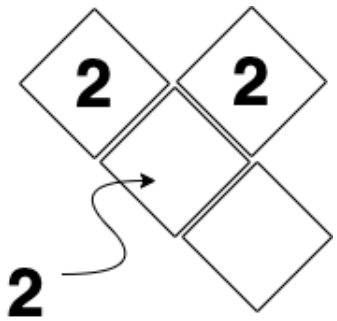
4. Decisión 2: Una carpeta contiene dos tipos de elementos: archivos y carpetas

Considere que usted debe modelar la funcionalidad de buscar todos los archivos que cumplen con cierta condición (predicado) en una estructura de directorios, donde dicha búsqueda se hace de manera recursiva, es decir, se busca en todos los directorios y subdirectorios. Construya un **Diagrama de Secuencia en UML** donde se evidencie la colaboración entre los objetos para cumplir con esta funcionalidad.

4 Pregunta 4: Genericidad, Reusabilidad y la Ortogonalidad en el desarrollo de software (7 puntos)

Uno de los tópicos más relevantes del curso de Desarrollo de Software, es la capacidad que los estudiantes deben desarrollar para identificar **problemas ortogonales** de manera que puedan hacer uso de técnicas de **reusabilidad** para aplicar el **Principio DRY (Don't Repeat Yourself)**. Dos técnicas de reusabilidad pueden ser las **clases genéricas (Genericidad)** y las **clases abstractas** (clases que forman una capa de **acoplamiento abstracto**) donde los subclases especializan el comportamiento implementando de manera específica los **métodos abstractos** de la clase base o padre.

Vamos a considerar un problema de la vida real, en particular, del dominio de los Videojuegos. En la siguiente figura se muestra una tablero parcial de casillas adyacentes (en este ejemplo son rombos).



El juego consiste en lo siguiente:

Desarrollo de Software
Primer Examen Parcial

1. Se dispone de un tablero con muchas casillas adyacentes, todas casillas de la misma forma. Pero, pueden existir en el juego tableros con casillas de otras formas: triángulos, cuadrados, rombos, hexágonos, etc.

2. En el tablero inicial del juego algunas casillas pueden tener valores pre-asignados. En este ejemplo hay números, pero podrían ser letras, colores, o cualquier otra cosa. Lo que es importante considerar es que un mismo tablero de juego solo puede tener valores del mismo tipo, es decir, si son números, son solo números.

3. El juego consiste en que el jugador vaya colocando valores en las casillas vacías, para ir logrando hacer un match de 3 o más casillas adyacentes.

4. Cuando se produce un match de 3 o más casillas adyacentes, se produce una unificación o mezcla (***merge***) de los valores y se genera un nuevo valor. El nuevo valor generado aparece en la casilla donde se colocó el valor inicial, y las otras casillas que hicieron match quedan limpias (sin valores).

En este juego se pueden identificar tres (3) problemas **ortogonales** entre sí: las reglas del juego, la forma de las casillas, los valores que van en las casillas.

Usted debe implementar completamente en **TypeScript/Dart/Java** la clase genérica abstracta **Casilla** que debe estar en función a dos tipos parametrizados **F** y **V**, donde el primero (la **F**) representa el problema ortogonal correspondiente a la forma de casilla y el segundo (la **V**) representa el tipo del valor que existe en una casilla. **Tengo en cuenta que la forma de la casilla, determina la relación de vecindad con sus casillas adyacentes.**

IMPORTANTE: Es **obligatorio** que usted implemente toda la lógica de juego descrita en la clase genérica abstracta **Casilla**. **NO** puede hacer uso de ninguna otra clase.

Tips: Puede hacer uso, en TypeScript, de arreglos, Map, tipos enumerados (enum).

5 BONUS (debe responder las 4 preguntas anteriores)

Explique cómo se relaciona el **Principio de Inversión de Dependencias** y la **Programación Orientada a Aspectos**. Justifique por qué ambos favorecen que el diseño sea flexible y más desacoplado.