Classes description:

Main -> this is the driver class or client

Minesweeper -> this is the class that creates an instance of the game, it is responsible about the logic behind the game.

Tile -> This class is an abstract class used to create the different buttons (tiles) for the game.

SafeTile -> this class is a concrete class of the Tile that represents a tile that has either a number (number of mines next to it) or nothing (there is no mines next to it).

MineTile -> this class is a concrete class of the Tile that represents a mine tile (the player should avoid).

TileFactory -> this is an abstract class that defines the blueprint for the different tiles factories.

SafeTileFactory -> this is a class that implements the TileFactory, it creates a SafeTile object.

MineTileFactory -> this is a class that implements the TileFactory, it creates a MineTile object.

TileState -> this is an interface that defines the methods the different tile states need to implement.

HiddenState -> this is a class that implements the TileState, it represents the initial state for the tiles, where the content is hidden.

FlaggedState -> this is a class that implements the TileState, it is used to represent a tile that have been flagged by the user (the user thinks it is a mine), the tiles content stays hidden.

RevealedState -> this is a class that implements the TileState, it represents the final state for the tiles, where the content is shown.

Design Patterns used:

(1) State: I used the state design pattern to separate the behavior of the Tile object depending on whether the object is revealed (should not be able to do anything), flagged (can remove the flag but not reveal it), and hidden (can flag or reveal).

Advantages:
(a) encapsulates the behavior of the tile in a state in a separate class. The alternative was having a Boolean to indicate if a tile is flagged, and another one for if it was revealed, and then checking these 2 Booleans before any step, so if it a stage one of the Booleans was not checked, a hard-to-find bug would occur.

(b) follows the open-close principle. adding a new state is now much easier and there would be no need to modify the existing code, only extend it for the new states.

Drawbacks:
(a) making the design more complex by increasing the number of classes

(b) currently there is only 3 states, and the behavior differences between them are small. So it might be an overkill.

____

(2) Singleton: this design pattern is used to ensure the user can only play once at a time. To do this, the constructer was made private, a static getInstance() method was created to access the single instance of Mincesweeper.

Advantages:
(a) Ensures that only one instance of the game exists, which avoids multiple game instances with shared resources.

(b) Provides a global access to the single instance of Minesweeper.

Drawbacks:
(a) difficult to test since the global state persists between tests.

____

(3) Factory: the Factory Method pattern is used to create Tile objects (SafeTile and MineTile). There is a TileFactory class which is an abstract class, and 2 concrete factory classes (SafeTileFactory and MineTileFactory). The concrete factory classes create their respective tiles.

Advantages:

(a) Encapsulates the object creation process, making it easier to maintain and modify.

(b) Follows the open-close principle, which makes it easier to add different types of tiles in the future without the need to modify the existing code.

Drawbacks:

(a) making the design more complex by increasing the number of classes