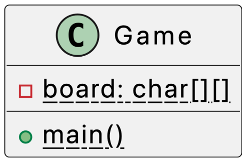
Programming Paradigms

Class Activity #13

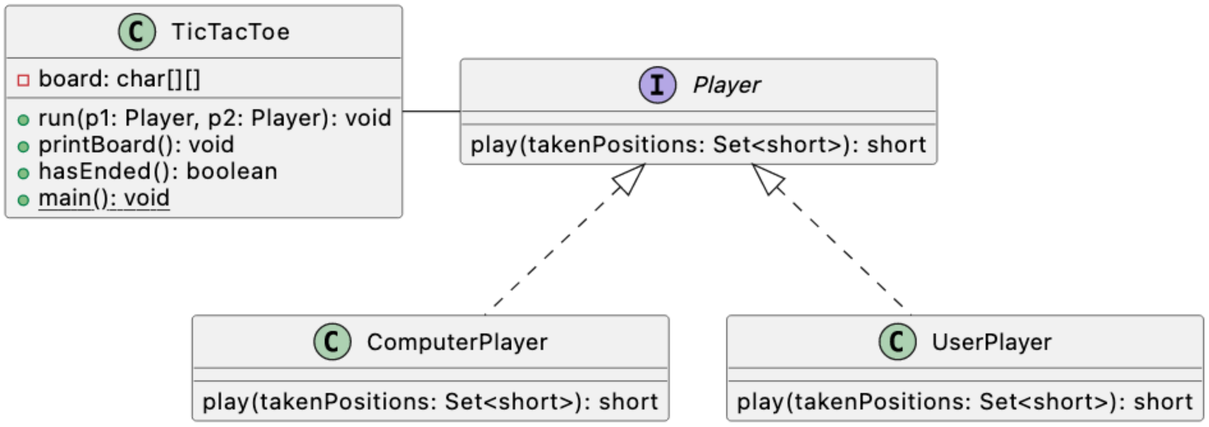
Design 1: Poor



There are many ways to solve this problem, one could simply have one single class (ex "Game") as shown in the left with the whole logic in it (inside main()). However, that's a bad idea and here's why: a single class is implementing different concerns (user player, computer player, managing the board, etc).

Design 2: Better

Instead, an alternative design is to have a player interface that dictates what a player can do (which is essentially picking a position in the board based on the current already taken positions). Then a **ComputerPlayer** implements that interface by randomly generating a position; and a **UserPlayer** class implements that interface by asking the user for a number. Then, we can have a class **TicTacToe** with a main method that basically creates an instance of **TicTacToe** and invokes the method **run(p1,p2)** such that **p1** is an instance of **UserPlayer** and **p2** an instance of **ComputerPlayer**.


TicTacToe
board: char[][]
run(p1: Player, p2: Player): void
printBoard(): void
hasEnded(): boolean
main(): void
Player
play(takenPositions: Set<short>): short
ComputerPlayer
play(takenPositions: Set<short>): short
UserPlayer
play(takenPositions: Set<short>): short

Notice how the design is now more modular. With it, it is easier for you to change the different players' logic. Say now that you have a "**SmartPlayer**" that relies on some fancy AI to pick the best positions; you could now create a class for it with the fancy AI logic, and then simply pass to the method **run(p1,p2)** from **TicTacToe** an instance of the SmartPlayer class. Since the method run(p1,p2) receives any object that implements the Player interface, that makes it easier to switch concrete player implementations.

Design 3: Best

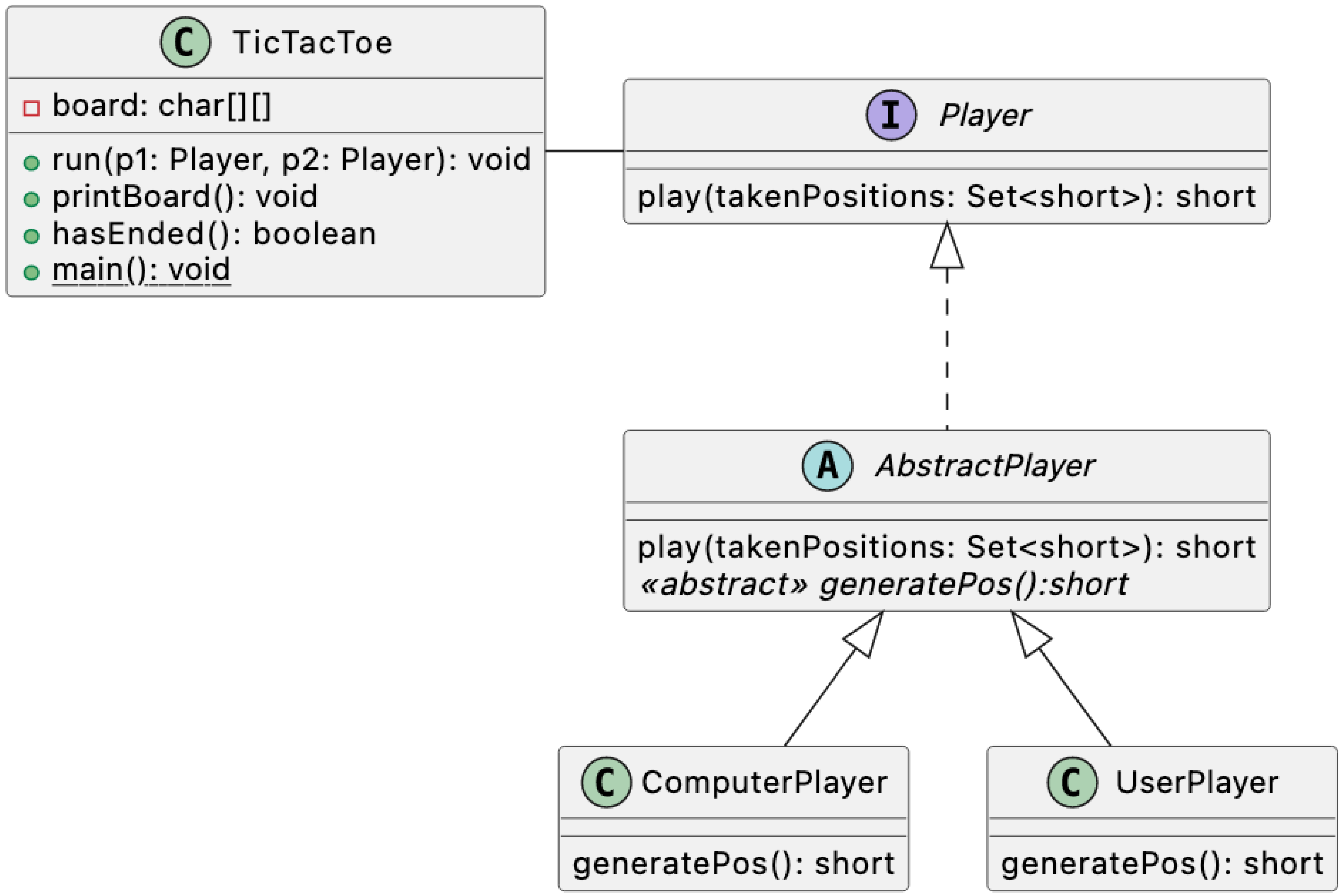
Notice that any player has to check whether a position is taken or not, before generating a position. The logic checking whether a position is taken or not is the same, regardless whether your player is a User, Computer, or SmartAI. Thus, we could change the design to use an abstract class:

- AbstractPlayer implements the play(Set) method as follows:

    It invokes the abstract method **generatePos()** (implemented by the subclass) to get a position;

    Checks whether the position is not taken; if it is taken, it goes back to the previous step.

    Once a valid position has been generated, it returns to the caller of the play method.


TicTacToe
board: char[][]
run(p1: Player, p2: Player): void
printBoard(): void
hasEnded(): boolean
main(): void
Player
play(takenPositions: Set<short>): short
AbstractPlayer
play(takenPositions: Set<short>): short
«abstract» generatePos():short
ComputerPlayer
generatePos(): short
UserPlayer
generatePos(): short