Lab 2 Task 2: Become familiar with class *NumberGame* Solution

2) Become familiar with class *NumberGame*

1. What attribute stores the players of the game?

   - The players of the game are stored in instance attribute *players*.

```
1    class NumberGame:
2        ...
3        def __init__(
4            self,
5            goal: int,
6            min_step: int,
7            max_step: int,
8            players: Tuple[Player, Player]
9        ) -> None:
10           ...
11           self.players = players # <- Here!
12
```

2. If *turn* is 15, whose turn is it?

   - We need to determine who's turn is at turn 15.

     The code of method *whose_turn* tells us

```
1        class NumberGame:
2            ...
3            def whose_turn(self, turn: int) -> Player:
4                """Return the Player whose turn it is on the given
     turn number.
5                """
6                if turn % 2 == 0:
7                    return self.players[0]
8                else:
9                    return self.players[1]
10
```

Using this code, we can conclude that at turn 15, it's player 2's turn.

3. Write a line of code that would create an instance of *NumberGame* that violates one of the representation invariants.

- We need write a line of code that violates one of the representational invariants.

  The representational invariant of the initializer of *NumberGame* tells us

```
1           """
2           ...
3           Precondition: 0 < min_step <= max_step <= goal
4           """
5
```

  It follows from this fact that the representational invariant is invalidated when $goal \leq 0$.

  Then, using this fact, we can write that a line of code that invalidates representational invariants is

```
1           NumberGame(-1,3,10,(Player(),Player()))
2
```

4. Which of the representation invariants is it possible to violate by constructing a *NumberGame* improperly?

- The following code tells us that the constructor of class *NumberGame* has four representational invariants as parameter types

```
1           def __init__(self, goal: int, min_step: int, max_step: int,
2               players: Tuple[Player, Player]) -> None:
3
```

  , and one as precondition

```
1           """
2           ...
3           Precondition: 0 < min_step <= max_step <= goal
4           """
5
```

  Using these facts, we can conclude that any of the five representational invariants can become violated when

  1. One or more arguments in *__init__* are of incorrect data type

2. $0 \geq min\_step > max\_step > goal$

5. List all the places in this class where a *Player* is stored, an instance attribute of *Player* is accessed or set, or a method is called on a *Player*

   - We need to find all places in *NumberGame* class where one a *Player* is stored, where an instance attribute of *Player* is accessed or set, or where a method is called on a *Player*.

   First, we need to find where *Player* is stored.

   By observation, we can conclude *Player* is stored in instance attribute *players* under the initializer method.

```python
class NumberGame:
    def __init__(self, goal: int, min_step: int, max_step: int,
                 players: Tuple[Player, Player]) -> None:
        """Initialize this NumberGame.

        Precondition: 0 < min_step <= max_step <= goal
        """

        self.players = players # Here
```

   Second, we need to find where the instance attribute of *Player* is accessed or set.

   By observation, we can conclude there are two places where one or more instance attributes of *Player* is accessed or set.

   The first one is inside *play* method.

```python
class NumberGame:
    ...
    def play(self) -> str:
        ...
        winner = self.whose_turn(self.turn - 1)
        return winner.name # <- Here!!
```

   The second one is inside *play_one_turn* method.

```python
class NumberGame:
    ...
    def play_one_turn(self) -> None:
        ...
        print(f'{next_player.name} moves {amount}.') # <-
    Here!!
        print(f'Total is now {self.current}.')
```

3

Finally, we need to find where method of *Player* is called.

By observation, we can conclude one method of *Player* is used, and it is called inside *play_one_turn* method.

```python
class NumberGame:
    ...
    def play_one_turn(self) -> None:
        next_player = self.whose_turn(self.turn)
        amount = next_player.move( # <- Here!!
            self.current,
            self.min_step,
            self.max_step,
            self.goal
        )

```