Learning Objectives - Mutability

- **Define the term mutable**
- **Construct a function to modify instance variables (attributes)**

Mutability and External Methods

Mutability

Objects are mutable, which means that objects (specifically their attributes) can change value. Think of a video game; the main character in the game is constantly changing. It could be their position on the screen, the score, their health, the items in their inventory, etc. Imagine a simple class called `Player`. A newly instantiated `Player` object has a `health` of 100, a `score` of 0, and starts on `level` 1. This object can lose health, increase their score, and advance levels.

```
//add class definitions below this line

class Player {
  int health;
  int score;
  int level;

  Player() {
    health = 100;
    score = 0;
    level = 1;
  }
}

//add class definitions above this line
```

Print out the attributes of `player1`. Then change each attribute and print out the attributes again.

```
  //add code below this line


  Player player1 = new Player();
  System.out.println("This player has " + player1.health + "
      health, a score of " + player1.score + " and is on level
      " + player1.level + ".");
  player1.health -= 10;
  player1.score += 25;
  player1.level += 1;
  System.out.println("This player has " + player1.health + "
      health, a score of " + player1.score + " and is on level
      " + player1.level + ".");


  //add code above this line
```

challenge

Try these variations:

- Change the health of `player1` to 0?
- Print the status of `player1` once their `health` is 0?
  - ▼ **One Possible Solution**

```
System.out.println("This player is dead. They died on
        level " + player1.level + " with a score of " +
        player1.score + ".");
```

External Methods and Objects

One of the benefits of methods is code reusability. The example above has a repetition of the `System.out.println` statement. This is a good opportunity to use a method.

```
  //add method definitions below this line


  public static void printPlayer(Player p) {
    System.out.println("This player has " + p.health + " health,
        a score of " + p.score + " and is on level " + p.level +
        ".");
  }


  //add method definitions above this line
```

In the `main` method, replace the strings inside the print statements with a call to the `printPlayer` method. Don't forget to pass the `player1` object to the `printPlayer` method.

```
  //add code below this line

  Player player1 = new Player();
  printPlayer(player1);
  player1.health -= 10;
  player1.score += 25;
  player1.level += 1;
  printPlayer(player1);

  //add code above this line
```

Using a method to print the status of `player1` may not seem like it was worth the effort to change the code. But when these methods become more complex, The efficiency becomes clear.

```
//add method definitions below this line

public static void printPlayer(Player p) {
  if (p.health <= 0) {
    System.out.println("This player is dead. They died on
      level " + p.level + " with a score of " + p.score +
      ".");
  } else {
    System.out.println("This player has " + p.health + "
      health, a score of " + p.score + " and is on level " +
      p.level + ".");
  }
}

//add method definitions above this line
```

Now that the `printPlayer` method will return two different strings, call the method when the `player1` object has full health, and call it again when the object has no health.

```
    //add code below this line

    Player player1 = new Player();
    printPlayer(player1);
    player1.health = 0;
    player1.score += 25;
    player1.level += 1;
    printPlayer(player1);

    //add code above this line
```

challenge

Can you:

- Create a function to change a player's health?
  ▼ **One possible solution**

```
public static void changeHealth(Player p, int amount) {
  p.health += amount;
}
```

- Create a function to change a player's level?
  ▼ **One possible solution**

```
public static void changeLevel(Player p) {
  p.level += 1;
}
```