# Lab 3 Examples: Adding and Removing Songs

**addSong(int index, Song song):** Add a new Song to a Playlist by implementing these steps:

1. Check whether the index is in the range from 0 to the first empty element (i.e., the last Song index plus 1). If the index is outside this range, do nothing.
   - Hint 1: Calculate the upper bound of the range using the variable numSongs.
   - Hint 2: Use a return statement to exit the method if the index is outside the range.
2. Check whether there is room in the array for the new Song. If not, create a new array that is twice as long and copy the Songs from the old to the new array.
   - Hint 1: Compare numSongs to the length to determine if the array is full.
   - Hint 2: Use the method copyOf from the Arrays class to create the new array.
   - Hint 3: Make sure to assign the new array to the variable songs.
3. Shift the existing Songs to make room for the new Song. Any Song with an index that is greater than or equal to the given index is moved to the next-highest index.
   - Hint 1: Use a for-loop.
   - Hint 2: Shift the last Song in the array first. That is, initialize the loop variable with the last Song index and decrement it after each iteration. (Why is this important?)
   - Hint 3: If the loop is written correctly, no Songs will be shifted when the given index is equal to the index of the first empty element.
4. Assign the new Song to the array element with the given index and increment numSongs.

To illustrate how the method works, consider the following code:

```
1. Playlist playlist = new Playlist();
2. playlist.addSong(1, song1);
3. playlist.addSong(0, song1);
4. playlist.addSong(0, song2);
5. playlist.addSong(0, song3);
```

**Line 1:** A new Playlist is created. The constructor creates a new Song array of length 2 and assigns it to the variable songs. The variable numSongs is initialized to 0. The state of the Playlist can be represented as follows:

`numSongs: 0`

| | |
|---|---|
| null | null |
| 0 | 1 |

**Line 2:** The index given to addSong is out of the valid range. The method does nothing, which leaves the Playlist unchanged:

`numSongs: 0`

| | |
|---|---|
| `null` | `null` |
| `0` | `1` |

**Line 3:** The given index is equal to the index of the first empty element. The array has enough room for the new Song and no existing Songs need to be shifted. The new Song is assigned to the specified element and numSongs is incremented:

`numSongs: 1`

| | |
|---|---|
| `song1` | `null` |
| `0` | `1` |

**Line 4:** The given index is equal to the index of an existing Song. The array has enough room for the new Song, but the existing Song must be shifted (i.e., copied from index 0 to index 1):

`numSongs: 1`

| | |
|---|---|
| `song1` | `song1` |
| `0` | `1` |

Now the new Song can be added:

`numSongs: 2`

| | |
|---|---|
| `song2` | `song1` |
| `0` | `1` |

**Line 5:** The given index is inside the valid range, but there is no room for the new Song. A new array is created with twice the capacity:

`numSongs: 2`

| | | | |
|---|---|---|---|
| `song1` | `song2` | `null` | `null` |
| `0` | `1` | `2` | `3` |

The existing Songs are shifted:

`numSongs: 2`

| | | | |
|---|---|---|---|
| `song1` | `song1` | `song2` | `null` |
| `0` | `1` | `2` | `3` |

Finally, the new Song is added:

`numSongs: 3`

| | | | |
|---|---|---|---|
| `song3` | `song1` | `song2` | `null` |
| `0` | `1` | `2` | `3` |

**removeSong(int index):** Remove and return a Song from a Playlist by implementing these steps:

1. Check whether the index is in the range from 0 to the last Song index. If the index is outside this range, return null.
   - Hint: Calculate the upper bound of the range using the variable numSongs.
2. Assign the Song at the given index to a temporary variable. (Why is this important?)
3. Shift the existing Songs to fill the gap left by the Song being removed. Any Song with an index greater than the given index is moved to the next-lowest index.
   - Hint 1: Use a for-loop.
   - Hint 2: Shift the Song to the right of the Song being removed first. That is, initialize the loop variable with the given index and increment it after each iteration. (Why is this important?)
   - Hint 3: If the loop is written correctly, no Songs will be shifted when the given index is equal to the index of the last Song.
4. Optionally, assign null to the last Song in the array.
   - Aside: This step is optional because the number of Songs is stored in the variable numSongs. However, setting the unused element to null allows removed Songs to be garbage collected more quickly.
5. Decrement numSongs.
6. Check whether both of the following conditions are true: (1) the number of Songs is less than or equal to 1/4 the array length, and (2) the array length is greater than the minimum capacity. If so, create a new array that is half as long and copy the Songs from the old to the new array. If the new length will be less than the minimum capacity, use the minimum capacity instead.
   - Hint 1: Use the method copyOf from the Arrays class to create the new array.
   - Hint 2: Make sure to assign the new array to the variable songs.
7. Return the removed Song. (You remembered to store it in a temporary variable, right?)

To illustrate how the method works, assume the code from earlier has already run, which produced a Playlist in the following state:

`numSongs: 3`

| | | | |
|---|---|---|---|
| song3 | song1 | song2 | null |
| 0 | 1 | 2 | 3 |

Now consider the following code:

```
1. playlist.removeSong(3);
2. playlist.removeSong(2);
3. playlist.removeSong(0);
4. playlist.removeSong(0);
```

**Line 1:** The index given to removeSong is out of the valid range. The method returns null and leaves the Playlist unchanged:

`numSongs: 3`

| | | | |
|---|---|---|---|
| song3 | song1 | song2 | null |
| 0 | 1 | 2 | 3 |

**Line 2:** The given index is equal to the index of the last Song (song2). No songs need to be shifted. Null is assigned to the element storing the last Song, and numSongs is reduced by 1:

`numSongs: 2`

| | | | |
|---|---|---|---|
| song3 | song1 | null | null |
| 0 | 1 | 2 | 3 |

The resulting array is more than 1/4 full, so a new array is not created. The removed Song (song2) is returned.

**Line 3:** The given index is equal to the index of the first Song (song3). The other Song (song1), which has a larger index, must be shifted to fill the gap (i.e., copied from index 1 to index 0):

`numSongs: 2`

| | | | |
|---|---|---|---|
| song1 | song1 | null | null |
| 0 | 1 | 2 | 3 |

Null is assigned to the element storing the last Song, and numSongs is decremented:

`numSongs: 1`

| | | | |
|---|---|---|---|
| song1 | null | null | null |
| 0 | 1 | 2 | 3 |

The array is 1/4 full and the length is greater than the minimum capacity. A new array is created with half the capacity:

`numSongs: 1`

| | |
|---|---|
| song1 | null |
| 0 | 1 |

The removed Song (song3) is returned.

**Line 4:** The given index is in the valid range. No songs need to be shifted. Null is assigned to the element storing the last Song, and numSongs is decremented:

`numSongs: 0`

| | |
|---|---|
| null | null |
| 0 | 1 |

The number of Songs is less than 1/4 the length, but the length is not greater than the minimum capacity. A new array is not created. The removed Song (song1) is returned.