**What are largest and secondLargest?**

* These are variables used to keep track of:
  1. **largest**: The biggest number in the array seen so far during the iteration.
  2. **secondLargest**: The second biggest number in the array seen so far during the iteration.

**What is Integer.MIN\_VALUE?**

* In Java, integers can have a range of values from **-2,147,483,648** to **2,147,483,647**.
* Integer.MIN\_VALUE is the smallest number an integer can store: **-2,147,483,648**.
* This value is chosen because:
  + It ensures that **any number** in the array will be larger than this initial value.
  + It avoids incorrectly assuming that the largest value starts at 0 or any other arbitrary value.

**Why Initialize largest and secondLargest to Integer.MIN\_VALUE?**

1. Imagine the array has **negative numbers**, like [-10, -20, -30].
   * Without initializing to Integer.MIN\_VALUE, the logic may fail because negative numbers are valid largest and second-largest numbers.
2. The program starts with no information about the array, so Integer.MIN\_VALUE acts as a "placeholder" that will be replaced as we process each element.

**Detailed Step-by-Step Example**

**Input Array: {12, 35, 1, 10, 34, 1}**

1. **Initialization**:
   * largest = Integer.MIN\_VALUE = -2,147,483,648
   * secondLargest = Integer.MIN\_VALUE = -2,147,483,648
   * These values will be replaced as we go through the array.

**Iteration 1: Process the first number, 12**

* Compare 12 with largest:
  + 12 > -2,147,483,648 → **Update largest to 12.**
* secondLargest remains -2,147,483,648 because there’s no second number yet.
* Now:
  + largest = 12
  + secondLargest = -2,147,483,648

**Iteration 2: Process the second number, 35**

* Compare 35 with largest:
  + 35 > 12 → **Update largest to 35.**
  + The previous largest (12) becomes secondLargest.
* Now:
  + largest = 35
  + secondLargest = 12

**Iteration 3: Process the third number, 1**

* Compare 1 with largest:
  + 1 < 35 → No change to largest.
* Compare 1 with secondLargest:
  + 1 < 12 → No change to secondLargest.
* Now:
  + largest = 35
  + secondLargest = 12

**Iteration 4: Process the fourth number, 10**

* Compare 10 with largest:
  + 10 < 35 → No change to largest.
* Compare 10 with secondLargest:
  + 10 < 12 → No change to secondLargest.
* Now:
  + largest = 35
  + secondLargest = 12

**Iteration 5: Process the fifth number, 34**

* Compare 34 with largest:
  + 34 < 35 → No change to largest.
* Compare 34 with secondLargest:
  + 34 > 12 → **Update secondLargest to 34.**
* Now:
  + largest = 35
  + secondLargest = 34

**Iteration 6: Process the sixth number, 1**

* Compare 1 with largest:
  + 1 < 35 → No change to largest.
* Compare 1 with secondLargest:
  + 1 < 34 → No change to secondLargest.
* Final Values:
  + largest = 35
  + secondLargest = 34

**Key Observations**

* At the end of the loop:
  + largest holds the biggest number: 35.
  + secondLargest holds the second biggest number: 34.

**Why Integer.MIN\_VALUE Works**

1. **Starting point**: Since all numbers in the array (12, 35, 1, 10, 34, 1) are greater than -2,147,483,648, the first number (12) replaces largest immediately.
2. **Updates secondLargest**: As the loop progresses, secondLargest is updated whenever a number is smaller than largest but larger than the current secondLargest.