LearnWithParth

Arrays in Java

LearnWithParth

Creating an Array in Java

Array in Java

- It can be of primitive type
  - E.g. array of integers, floats, boolean etc.
- It can be of Class type
  - E.g. array of String, Demo, Buttons etc.

Creating an Array

- Declare
  - Assigning type of the array

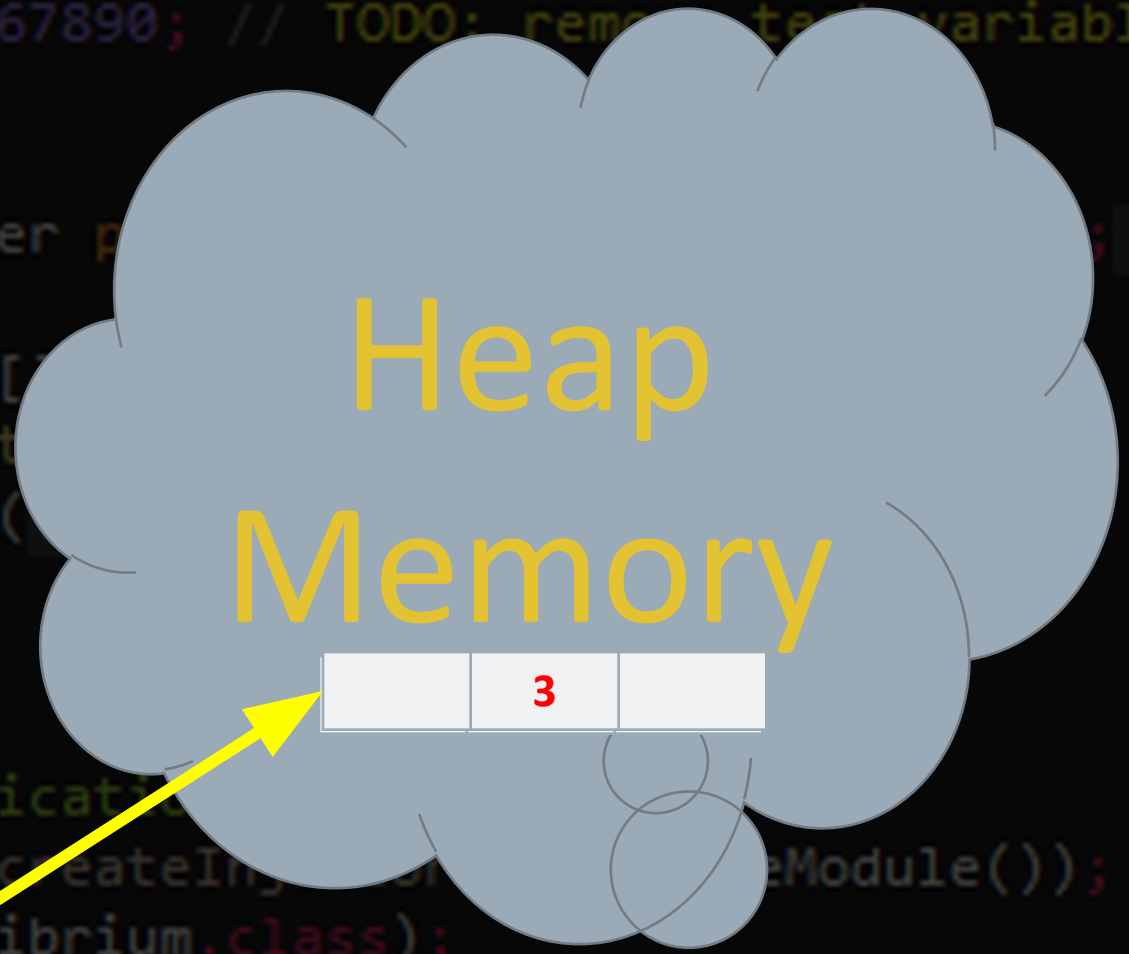
- Construct
  - Assigning size of the array

- Initialize
  - Assigning values to the elements of the array

LearnWithParth

Creating an Array

- Declare
  - int a[];
- Construct
  - a = new [3];
- Initialize
  - a[1]= 3;



LearnWithParth

Different ways to declare the array

- int values [];
- int [] values;
- String names [];
- String [] names;
- int[5] scores; // will NOT compile

LearnWithParth

Different ways to construct the array

```
int array1 []; array1 = new int[4];

int[] array2 = new int[4];

String names[] = new String[10];

int[] carList = new int[]; // Will not compile; needs a size
```

LearnWithParth

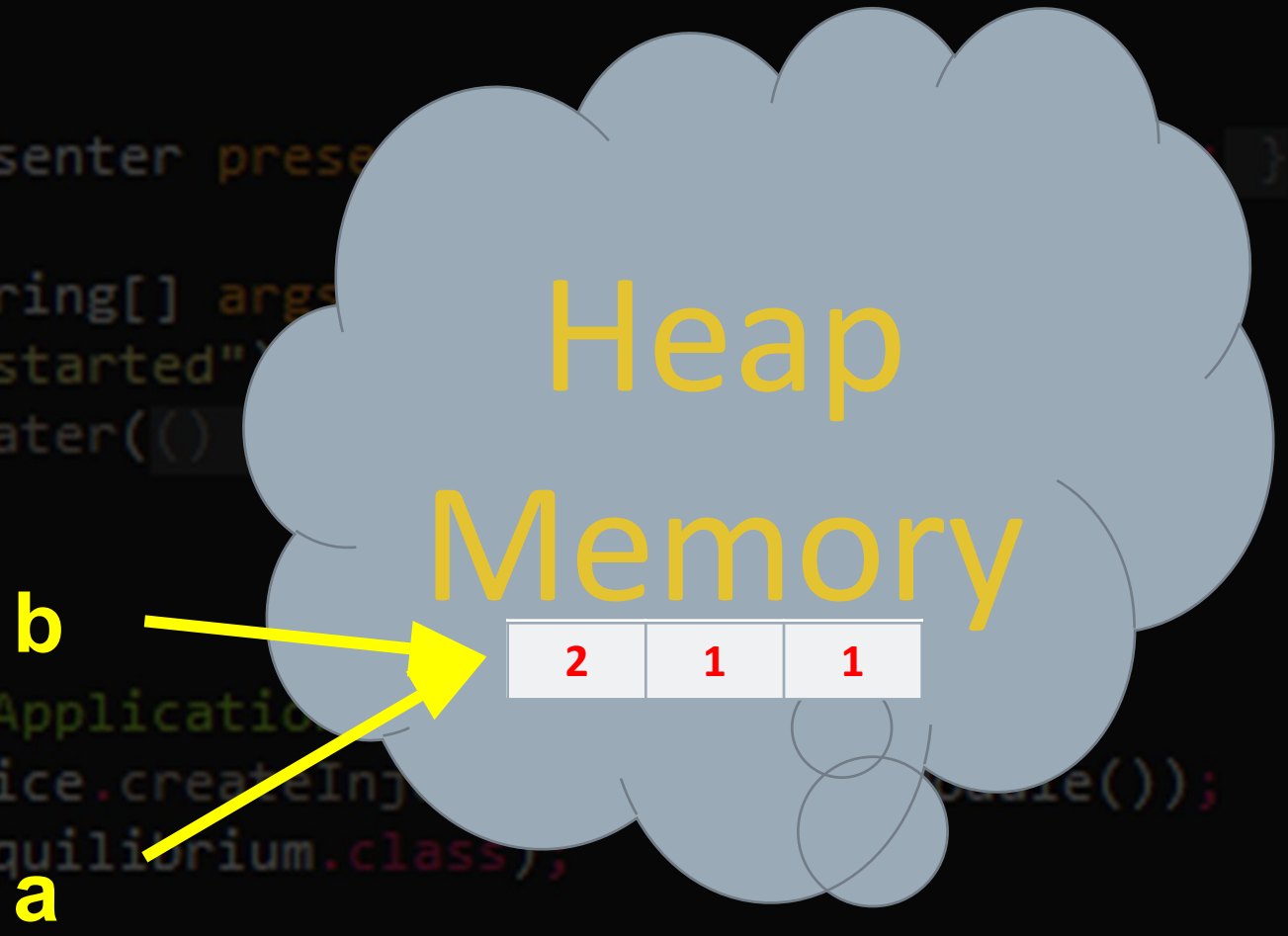
Different ways to initialize the array

```
int array[] = new int[2]; array[0] = 1; array[1] = 2;

int [][] vector = {{5, 2, 4, 7}, {8, 2}, {1}};

int[] values = new int[] {6, 3, 1};

int array2[] = {2 ,3 ,4 ,5}; int bArray[] = array;

int [] a, b; // or int a[], b[];
```

LearnWithParth

Array with multiple references

- int a[] = new [3];
- a[0]=1, a[0]=1, a[0]=1;
- int b[] = a;
- b[0] = 2;




DEMO


QUIZ
TIME

Arrays Class

Arrays Class

- Class Hierarchy:
  - java.lang.Object
  - ↳ java.util.Arrays
- Class Declaration:
  - public class Arrays
  - extends Object
- Syntax to use Array:
  - Arrays.<function name>;

Example

```
int intArr[] = {10, 20, 15, 22, 35};
Arrays.sort(intArr); // [10, 15, 20, 22, 35]
int intKey = 10;
Arrays.binarySearch(intArr, intKey); // 10 found at index = 0
Arrays.binarySearch(intArr, 1, 3, intKey); // 10 found at index = -2
int intArr1[] = {10, 15, 22};
Arrays.equals(intArr, intArr1); // Integer Arrays on comparison: true
int intArr2[] = Arrays.copyOf(intArr, 10); // [10, 15, 20, 22, 35, 0, 0, 0, 0,
0]
int intArr3[] = Arrays.copyOfRange(intArr, 1, 3); // [15, 20]
```

LearnWithParth

ArrayList Class

ArrayList Class

- Class Hierarchy:
  - java.lang.Object
  - ↳ java.util.Collection
  - ↳ java.util.List
  - ↳ java.util.ArrayList
- Class Declaration:
  - public class ArrayList
  - implements List extends Collection
- Syntax to use ArrayList:
  - ArrayList list=new ArrayList();

Example

- ArrayList<String> list=new ArrayList<String>();//Creating arraylist
- list.add("Mango");//Adding object in arraylist
- list.add("Apple"); list.add("Banana"); list.add("Grapes");
- System.out.println(list);
- Iterator itr=list.iterator(); //getting the Iterator
- while(itr.hasNext()){ //check if iterator has the elements
- System.out.println(itr.next()); //printing the element and move to next
- for(String fruit:list)
- System.out.println(fruit);
- list.set(1,"Dates");
- Collections.sort(list);

Iterating ArrayList

```
List<Integer> numbers = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8));
```

```
// Iterating using for loop
for (int i = 0; i < numbers.size(); i++)
    System.out.print(numbers.get(i) + " ");
```

```
// For each loop
for(Integer var : numbers)
    System.out.println(var);
```

```
// Using iterator
Iterator iterator = numbers.listIterator();
while(iterator.hasNext())
    System.out.println(iterator.next());
```

```
// Iterating using while loop
int val = 0; int i = 0;
while(numbers.size()>val){
    System.out.print(numbers.get(i) + " ");
    val++;
}
```

```
// Using lambda function
numbers.forEach(number->System.out.println(number));
```

```
// Using enumeration
Enumeration<Integer> e = Collections.enumeration(numbers);
while(e.hasMoreElements())
    System.out.println(e.nextElement());
```

What Do You Do with a Collection?

- Add objects to the collection.
- Remove objects from the collection.
- Find out if an object (or group of objects) is in the collection.
- Retrieve an object from the collection without removing it.
- Iterate through the collection, looking at each element (object) one after another.

Advantages of ArrayList over array

- It can grow dynamically.
- It provides more powerful insertion and search mechanisms than arrays.


Questions?