Module 1-7

Lists, Stacks, and Queues

Java Collections

Collections offer an alternative to storing objects (They are array-like).

Lists

- Lists can store objects, and behave in a similar fashion as arrays.
- Lists are zero-based just like arrays.
- Unlike arrays, the list’s size does not need to be predefined.

Lists: Declaring

- We will be using the following List declaration:

```
List <Type> name = new ArrayList<Type>();
```

- Here is an example of a list containing Strings:

```
List <String> strList = new ArrayList<String>();
```

- You cannot declare Strings with primitive data types, if you want say a list of integers you must use the wrapper instead:

```
List<Integer> intList = new ArrayList<Integer>();
```

List Methods: add & get

- The add() method puts a new item inside the list.

```
List <String> myCats = new ArrayList<String>();

myCats.add("FluffBall");
myCats.add("Rambo");
myCats.add("Zoltan");
```

- The get() method retrieves an item from the list. We pass to this method an index, continuing the above example:

```
String myFavCat = myCats.get(0);
System.out.println(myFavCat);
// prints out FluffBall
```

List Methods: size

- The size() method returns the size of the list.

```
myCats.add("FluffBall");
myCats.add("Rambo");
myCats.add("Zoltan");

System.out.println(myCats.size());
// prints 3
myCats.add("PrincessJava");

System.out.println(myCats.size());
// prints 4
```

List Methods: remove

- We can remove a list item by using the remove method and specifying the index of the item that we want to remove:

```
List <String> myCats = new ArrayList<String>();

myCats.add("FluffBall");
myCats.add("Rambo");
myCats.add("Zoltan");

myCats.remove(1);
// Rambo has been removed from the list.

System.out.println(myCats.get(1));
// Will now print Zoltan
```

The For Each Loop

Now is a good time to introduce a condensed version of the for loop that is helpful for iterating through lists. The for each loop has the following format:

```
for ( data type placeholder name : name of array or list ) {

    ...

}
```

To the left of the colon we specify a placeholder variable, in each iteration of the loop, we can access the current value of the array or list by simply using the placeholder variable.

Let’s work with some lists!

Stacks

- Stacks can also store objects.
- It specifically does so in a **LIFO** manner (**L**ast **I**n **F**irst **O**ut).
- Think of a stack of plates:

- The last plate was placed last onto the stack.
- If you retrieve a plate, you will retrieve the one that was placed last.

Stacks: Declaring

- We will use the following declaration for declaring a stack:

  ```
  Stack <Data Type> name = new Stack<Data Type>();
  ```

- Here is an example:

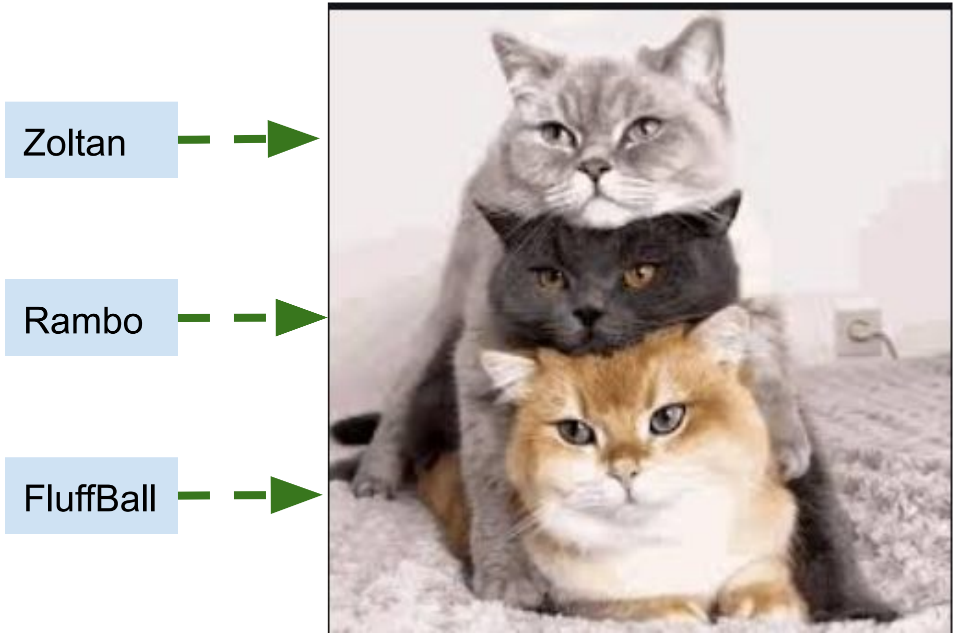
  ```
  Stack <String> thisStack = new Stack <String>();
  ```

Stacks Methods: push

- The push method adds an item to the stack:

```
Stack<String> catStack = new Stack<String> ();

catStack.push("FluffBall");
catStack.push("Rambo");
catStack.push("Zoltan");
```



Now if you want to pick up the cats, you would logically want to pick the kitties from the top starting from Zoltan (otherwise chaos would ensue), which leads us to our next method...

Stacks Methods: pop

- The pop() method removes the item from the stack that was

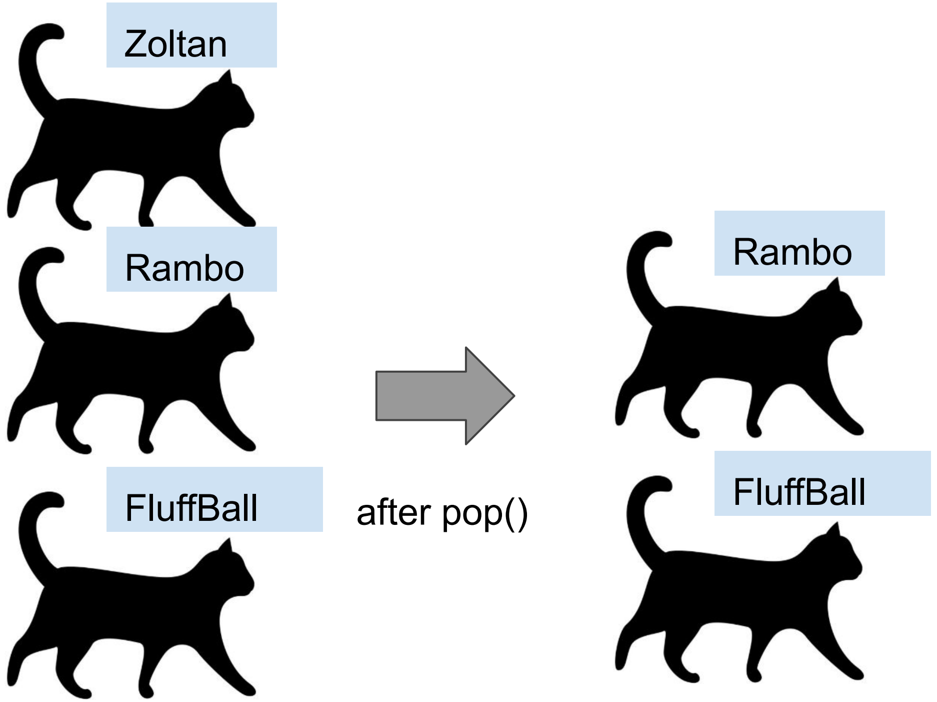
```
Stack<String> catStack = new Stack<String> ();

catStack.push("FluffBall");
catStack.push("Rambo");
catStack.push("Zoltan");

for (String cat : catStack) {
        System.out.println(cat);
        // FluffBall, Rambo, Zoltan
}

catStack.pop();

for (String cat : catStack) {
        System.out.println(cat);
        // FluffBall, Rambo
}
```



Stack Methods: peek

- The peek() method allows you to get the top most item in a stack without deleting it.

```
Stack<String> catStack = new Stack<String> ();

catStack.push("FluffBall");
catStack.push("Rambo");
catStack.push("Zoltan");

String topCat = catStack.peek();
System.out.println("Uh oh, " + topCat + " is being a meanie again!");

for (String cat : catStack) {
	System.out.println(cat); // Still prints all 3 cats.
}
```

The first println will print: “Uh oh, Zoltan is being a meanie again!”

Let’s work with some stacks!

Queues

- Queues can also be used for storage.
- It specifically does so in a **FIFO** manner (**F**irst **I**n **F**irst **O**ut).
- Think of a line at a store checkout counter:

Arrived last, so will be served last.

Arrived first, so will be served first.

Queues: Declaring

- The following syntax is used to declare a queue:

  Queue <**Data Type**> name = new LinkedList<**Data Type**>();

- Here is an example:

  ```
  Queue <String> trainedCats = new LinkedList<String>();
  ```

Queue Methods: offer

- We use the offermethod to put items in a queue:

```
Queue<String> trainedCats = new LinkedList<String> ();

trainedCats.offer("FluffBall");
trainedCats.offer("Rambo");
trainedCats.offer("Zoltan");


for (String cat : trainedCats) {
	System.out.println(cat);
}
```

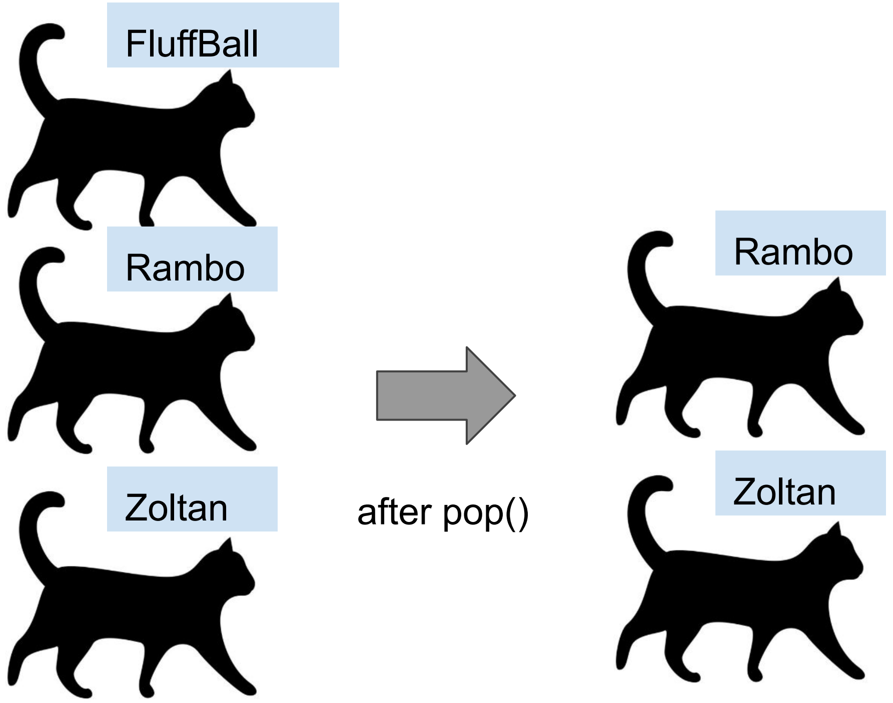
Queue Methods: remove

The remove() method takes an item out from the front of the line.

```
trainedCats.offer("FluffBall");
trainedCats.offer("Rambo");
trainedCats.offer("Zoltan");

trainedCats.remove();

for (String cat : trainedCats) {
        System.out.println(cat);
}
// prints out Rambo Zoltan
```



Queue Methods: peek

- The peek method, like with stacks, allows you to look at the first element of the list without removing it.

```
Queue<String> trainedCats = new LinkedList<String> ();

trainedCats.offer("FluffBall");
trainedCats.offer("Rambo");
trainedCats.offer("Zoltan");

String firstCat = trainedCats.peek();

System.out.println(firstCat + " is up early");
```

- “FluffBall” will be assigned to the String firstCat.

Let’s work with some queues!