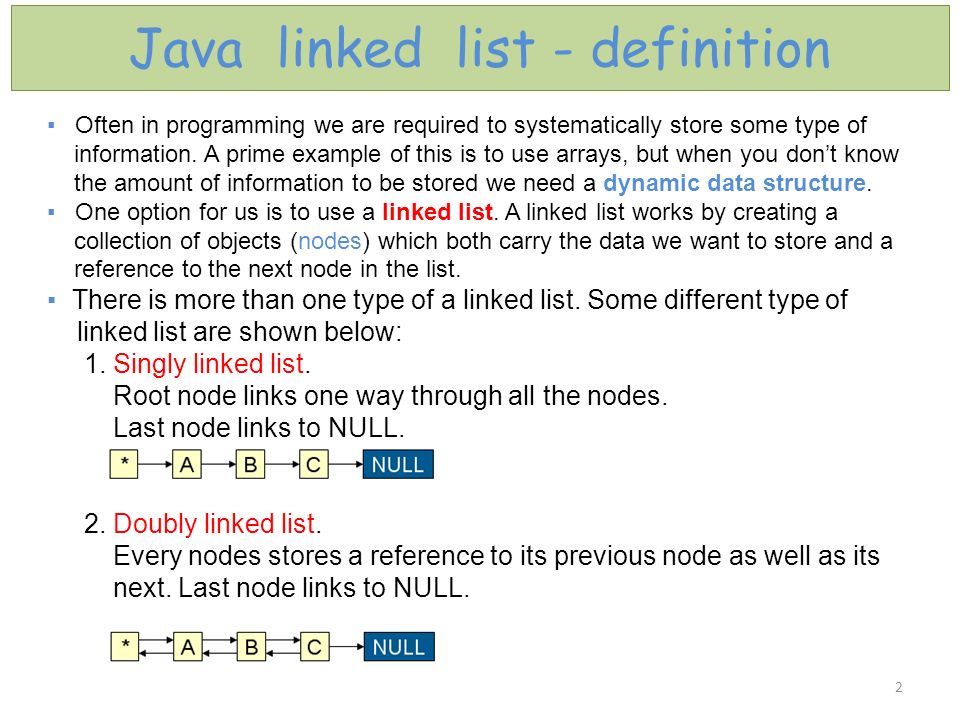
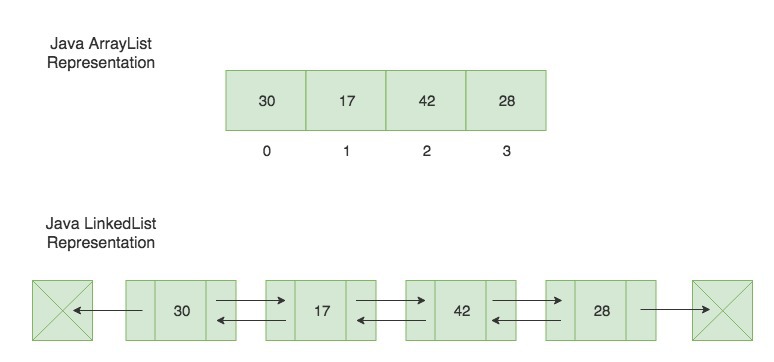
# **Linked List**





public class LinkedList {

private Node first, last;

//Declares private variables `first` and `last`, which are //instances of Node objects, to keep track of the first and last

//nodes of the list.

}

//LinkedList => keeps track of the First and last entre into a list

import java.util.\*;

LinkedList<String> animalList = new LinkedList<String>();//constructor

animalList.add("frog");

animalList.add("giraffe");

animalList.add("buffalo");

animalList.add("mongoose");

animalList.addFirst("shark");

]//Puts shark as the first element in the ArrayList

animalList.addLast("koala");

//Puts koala as the last element in the ArrayList

animalList.add(3, "cuttlefish");

//This adds the value cuttlefish to the 3rd index of the LinkedList

animalList.remove(3);

//Removes the element that is at the 3rd index of the LinkedList

animalList.remove("piranha");

//Removes the first instance of "piranha" in the LinkedList

animalList.removeFirst();

//Removes the first element of the LinkedList

animalList.removeLast();

//Removes the last elements of the LinkedList

animalList.contains("giraffe");

//Returns `true` if "giraffe" is present in the LinkedList

animalList.size();

//Returns an integer denoting the size of the LinkedList

animalList.get(3);

//Returns the value located at the 3rd index of the LinkedList

animalList.set(4, "emu");

//Sets the value at index 4 to "emu"

//Queue => acts as an interface, First in, First Out

Queue<Integer> myQueue = new LinkedList<>();

// Adds elements {5, 6, 7, 8, 9} to queue

for (int i=5; i<10; i++)

myQueue.add(i);

// Display contents of the queue.

System.out.println("Elements of queue-"+ myQueue);

// To remove the head of queue.

int removedHead = myQueue.remove();

System.out.println("removed element-" + removedHead);

System.out.println(myQueue);

// To view the head of queue

int head = myQueue.peek();

System.out.println("head of queue-" + head);

// Rest all methods of collection interface,

// Like size and contains can be used with this

// implementation.

int size = myQueue.size();

System.out.println("Size of queue-" + size);

// Stacks => Last in, first out

import java.util.\*;

//Imports the data we need to use the Stack class

Stack<Integer> myStack;

//creates a reference variable for a Stack called myStack

myStack = new Stack<Integer>();

//creates a new Integer Stack and assigns its address to myStack

myStack.push(1);

//Pushes 1 to the top of our stack

//myStack now consists of [1]

myStack.push(2);

myStack.pop();

//returns 3 and removes it from the stack

//myStack now consists of [1, 2]

//We can also look at the top of the stack without

//removing it by using peek

myStack.peek();

//returns 2, but doesn't remove it

//myStack still consists of [1, 2]

//We may want to know if a stack is empty before we manipulate it:

myStack.empty();

//this will return `false` because our stack is not empty

//myStack now consists of [1, 2]

myStack.push(3);

//myStack now consists of [1, 2, 3]

//Despite pushing to the "top" of the stack, the printed representation

//puts our new numbers at the back. The important part is that they are

//removed from the same place they are added.

# **Arrays**

An array can hold any sort of data type.  In Java, **all elements in an array must be of the same data type**.

Just like characters in Strings, items in an Array can be called by their index:

String[] my\_array = {"mary","sybil","edith"};

console.printf(my\_array[1]);

// returns "sybil"

Arrays are part classs is part of the java.util package and can bring in the inheritance of the Aray class that let us create nad inspect our array.

To find the size of the array, we call the **.length**method**.**

While Arrays are fixed sized**, ArrayLists**are dynamic sized arrays in Java that allow us to add and remove elements.

To modify our ArrayList, we call on the methods associated with the ArrayList object - **.add(), .get(), .remove()**

ArrayList<String> names = ArrayList<String>();

//Creates an array of string called "names"

ArrayList<String> names = new ArrayList<String>(5);

//Creates an array of string called "names" that has 5 indexes

//The size is not permanent and can grow or shrink.

//adds elements to the end of the array

names.add("John");

names.add("Heather");

names.add("Joe");

//adds an element to a specifc index

names.add(2, "Finn"); //adds "Finn" to index 2, moves "Joe" to index 3

//retrieves element values based on index number

names.get(1); //returns "Heather"

//removes an element from an ArrayList

names.remove("Finn"); //takes "Finn" out of the list, moves "Joe" back to index 2

// index 3 no longer exists

names = {John, Heather, Joe, Finn} //original ArrayList

//reverse the contents by calling Collections.reverse(ArrayListName)

Collections.reverse(names);

=>[Finn, Joe, Heather, John] //after reversing the elements

//Get the size of the ArrayList

System.out.println(names.size());

=> 4

//Checks the array list for a specific element

names.contains("Heather")

=> true

//remove all elements from list

names.clear();

=> names ={}

Sometimes, it may be easier to think of an Array as a String, or vice versa.  For conversion from a String to an Array, we use the String method **.split()**.

//Create a new string

String myString = "I really love icecream!";

//Create another string that holds the split version of the first

String str[] = myString.split(" ");

//Use the List class to create a new ArrayList

List<String> arrayString = new ArrayList<String>();

//Store the contents of our split string as elements in

//arrayString using the Array.asList to convert the string into an array

arrayString = Arrays.asList(str);

=>[I, really, love, icecream!] //Turns the string myString into an array

To go the other way, it's a little easier.  We use the String class method **.toString()**

//ArrayList of strings

fruitList = {"Apple", "Banana", "Grape"}

//Convert the list to an Array using .toArray method

Object[] fruitArray = fruitList.toArray();

//create a new string to hold the converted array in string form.

String fruitString = Arrays.toString(fruitArray);

=>[Apple, Banana, Grape] //output of fruitString

//to remove brackets and commas, use the String method .replace

System.out.println(Arrays.toString(fruitArray).replace("[","").replace("]","").replace(",", " "));

=> Apple Banana Grape

# **HashMaps**

While Arrays are stored by index, HashMaps store data as "key"/"value" pairs.

HashMap<String, Integer> person = new HashMap<>();

//.put adds items to the hash map

person.put("Joan", 21);

person.put("Anna", 34);

person.put("Mike", 55);

person.put("Daniel", 42);

//creates the following HashMap

=> {Joan=21, Mike=55, Daniel=42, Anna=34}

//.get retrieves data from the hash map

person.get("Joan");

=> 21 //returns the value associated with the key

//.remove takes the specified key and it's value out of the hash map

person.remove("Mike");

=>{Joan=21, Daniel=42, Anna=34}

//replace the value of a given key

person.put("Joan", 42);

=>{Joan=42, Daniel=42, Anna=34} //changes the associated value from 21 to 42

//checks for a key within a HashMap. Returns a boolean value

person.containsKey("Daniel");

=>true

//checks for a key within a HashMap. Returns a boolean value

person.containsValue(34);

=>true

//Returns true if this map contains no key-value mappings.

person.isEmpty();

=> false

# **Loops**

## **For Loops**

Executes a task a fixed number of times

for (int num = 0; num < 10; num++ ){

System.out.println(num);

}

 the **index** is **num**and is initialized to 0

 the **condition** is checking for num less than 10,

 the  **num++** increases the value of **num** through each pass through the loop.

 The line within the braces is the task that is performed if the condition evaluates to TRUE

## **While Loops**

The **while** loop tests a condition and performs the tasks in the braces if the condition is true

While we loop, we increment the variable, so that 'count' will eventually equal 5.

Otherwise, we would create an **infinite loop**, and it will just be printing "0" forever.

int count = 0;

while (count < 5){

System.out.println(count);

count ++;

}

//returns the output

0

1

2

3

4

## **Do-while Loops**

This loop is similar to the do Loop, except the conditional goes in a different place.

count = 10;

int limit = 0;

do{

System.out.println("Hello");

limit ++;

} while (limit < count); count = 10;

int limit = 0;

do{

System.out.println("Hello");

limit ++;

} while (limit < count);

The do-while loop does the conditional test after the statement inside the loop, rather than before.  Just like the while loop, you need to change the value of the index to prevent an infinite loop.

## **Exiting a Loop**

Exiting a loop usually happens once the condition evaluates to false.  Sometimes you may want to exit a loop immediately, even if the condition being evaluated is true.  You can do this using **break** and **continue**.

int index = 0;

while (index <= 500){

index +=5;

if (index == 350){

break;

}

}

**break** exits the loop that contains it.

int index = 0;

while (index <= 500){

index +=5;

if (index == 350){

continue;

}

}

continue causes the loop to exit its current pass through the loop and starts from the beginning

## **ArrayLists forEach**

Here is how a forEach loop through an ArrayList is formed:

fruitList = ["Apple", "Banana", "Grapes"]

for (String fruit : fruitList){

System.out.println(fruit);

}

$ java ArrayPractice

Apple

Banana

Grapes

where **fruit** represents each element in the **fruitList**array list.

Each array element is print with each loop through the array list.

## **HashMap Map.Entry**

person = {Joan=22, Daniel=42, Anna=34}

//loop through the hash map and return each key/value pair

for (Map.Entry<String, Integer> entry : person.entrySet()){

String key = entry.getKey();

Integer value = entry.getValue();

System.out.println(key + ":" + value);

}

Map.Entry interface allows us to work with a hash map entry.

The entrySet( ) method returns a Set containing the map entries. Each of these set elements is a Map.Entry object.

$ java HashPractice

Joan:22

Daniel:42

Anna:34

\*Ignore the coloring, sorry!