**Encapsulation:**

Hiding the information and controlling the access of the data

Ex:bank transactions,passwords etc…

DTO -data transfer object-🡪properties should be private,must have an no arg constructor,must have setter and get method,toString method,equals method.

EX: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/Access%20Specifier>

**Abstraction:**

Knowing the functionality but don’t know the implementation.

Incomplete idea

Ex: we have a thoughts to buy an apartment but physically do not exist. planning to have.

Abstract 🡪 it is a keyword, it means incomplete idea. Which is used in declaring class and methods.

When a class have an abstract method that class should be a abstract class.

Ex:Number,httpservlet,generalservlet. These are the exist abstract class

The abstract class can only get by inheriting it.

Ex: public abstract class demo{

Public abstract void display();

}

Public class child extends demo{

@override

Public void display(){

Sout(“displaying”);

}

EX: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/Product/src/com/xworkz/product>

**Exception**:

Exception is a exceptional event which is a scenario that will distrubs the normal or regular flow.

Exception implements the Throwable interface.

Throwable is a interface is used handle the exceptions.

Throwable has two sub interfaces--🡪 Error and Exception.

Throwable

Runtime Exception

Exception

Error

Program issue

System issue

Data issue

Logical issue

Here Errors are used for the system issue.

Exception is used for the Program issue but in Exception it will deal with only data issue for logical issue RuntimeException will handle it. In a class if Exception or its sub interfaces are created the compiler will force to handle the event or exception is known as checked or compiletimeException.

RuntimeException is sub interface of Exception it has also so may sub interfaces. When RuntimeException or it sub interface are occurred the complier not force to handle it is also called as uncheckedException or RuntimeException.

There are 5 keywords are there

* Throw🡪 Create an event
* Throws🡪delegate the event
* Try🡪absorb the event
* Cath🡪handle the event
* Finally🡪handle the resources

Throw: Throw keyword is used to create an event. If know the possible exception the we can create the event to tell that jvm to stop the execution.

When throw keyword is encountered the execution will stop and look for the handler if handler was not found then execution will be terminated.

Throws: Throws keyword is used for delegate the event means if we don’t know how to handle the event the we can tell that jvm to handle it own or to do some one.

Try: try block is a special block which absorbs the event means when we create an event inside the try block it will stop the execution and look for the handler if handler is not there it comes out of the try block only. It will not terminate the whole program.

Try can used with the diff handlers.

* Try with catch

Example of try

<https://github.com/deepthins08/Core_Java/tree/main/Intellij/Product/src/com/xworkz/product/exception/blocks>

* Try with multiple catch
* Try with multi catch
* Try with resources
* Try with finally
* Try with multiple catch and finally
* Try with multi catch and finally
* Try with catch and finally

Catch: it is used to handle the event or catch the event.

Syntax is

catch (subinterfaceException e) {

e.message();

}

Finally: it is used to handle the resources means used to close the events. Explicitly we call close() in finally.

When we used try with resources it will have automatic close method in it.

**Collections: Collection API (Application programming interface):** which is used to connect to the classes.

Collection is set of interfaces, classes, abstract classes, enum classes, annotations.

Collections are grouping elements it is overcome the drawback of arrays. Array is fixed size we can’t modify the length or size of the array. But in collection we can add n number of elements to the array.

Collection is in the package of java.util

List and set are sub interfaces of collection

Iterable

TreeSet

HashSet

LinkedHashSet

LinkedList

ArrayList

Set

List

Collection

Dotted lines are representing classes that are implemented those interfaces.

The methods of this class all throw a NullPointerException if the collections or class objects provided to them are null.

Example for collection:

<https://github.com/deepthins08/Core_Java/tree/main/Intellij/collection/src/com/xworkz/collection>

**Comparator:**

**Comparator** which is used in collections for sorting the data in ascending and descending order. The comparator interface had a method compare method which will do the all the logic to sort the data. In order to sort the data in ascending and descending order we have to override the method. Using comparator we can sort all the properties by ascending and descending order.

Ex: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/customDataTypeCollection/src/com/xworkz/customtype>

Ex 2: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/collection/src/com/xworkz/collection>

**Comparable:**

**Comparable** which is used in collection in order to sorting the data. Using comparable we can set default property to sort in ascending or descending that can’t be changed by overriding the **compareTo** method.

Ex: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/comparable/src/com/xworkz/comparable>

**Lambda Expression:**

**Lambda expression** is used to sort the data in collection by reducing the code. If try to sort the properties in comparator or comparable we have to write separate class for to override the method for each property. but if we use lambda expression, we don’t have to write another class we can sort directly using lambda expression.

Syntax: Comparator<> comp=(parameters) -> {

Statement;

}

Comparable<> comp1=(param)-> {

}

Ex: <https://github.com/deepthins08/Core_Java/tree/main/Intellij/lamdaExpression/src/com/xworkz/lamdaExpression>

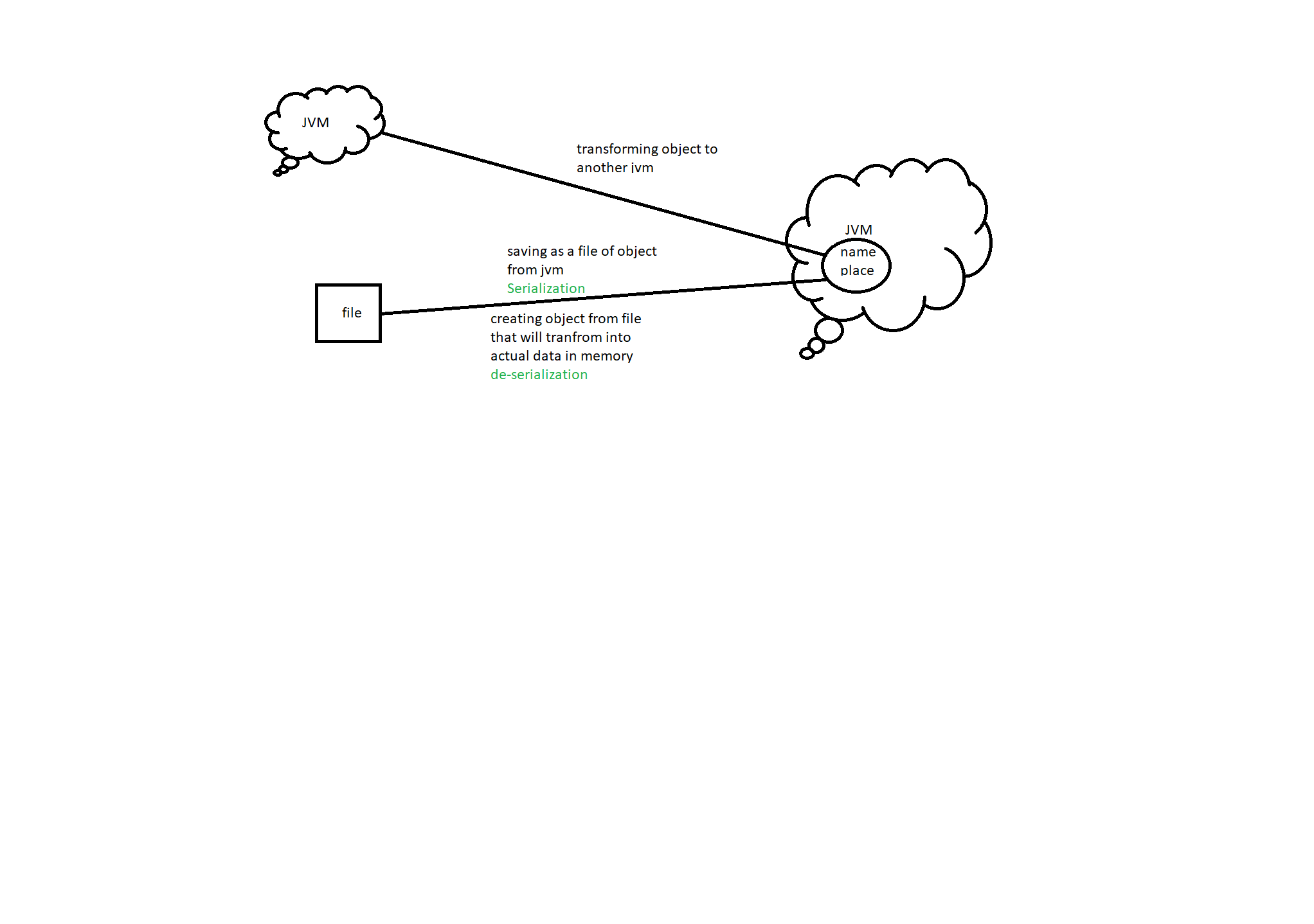
🡪

We can create an object in 4 ways

* New
* New instance
* Serializable
* clone

Serializable🡪

Serializable is a interface which is implememted in classes to save the object into file or transfer the object into another jvm. Saving object to file is known as serialization. And creating object from the file and get back to the class is known as de-serialization.



When we used serializable in a class means that in future anybody want save the code in a file or transfer to another jvm by implementing serializable it denotes that the class is ready to save or transfer.

**ArrayList:->**

ArrayList in collection is dynamic but it is also an static array but in collection it makes it as dynamic.

System.arraycopy() this method will used in ArrayList.

In ArraList which consist Object[] it is fixed as array size to 10

Object[] obj=new Object[10];

ArrayList<String> list=new ArraList<>();

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

List.add(“abc”);

The list is added to 10 values to the array

When we add the data to the list it will create new array with more size and copies all elements from the first array.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  | 10 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  | 15 |

The first array is will be removed.

Likewise if add more elements to the list it will create new array and copies the elements to the new array and adds the new elements to that array.

If we delete or remove an element it will create new array that will decrease it size and copies the elements to that array.

When we use add() method which add it sequentially.

If we use add(3,”abc”);

It will add abc at index 3 and it moves the other data to next index. It will move every element to next index so it makes it slow in insertion or in deletion. But searching is fast in ArrayList because of its insertion order.

**LinkedList:**

Linked list it will have nodes in it that means it have address of previous element and address of next element.

|  |  |  |
| --- | --- | --- |
| Address of Previous node | element | Address of next node |

LinkedList<String> list=new LinkedList<>();

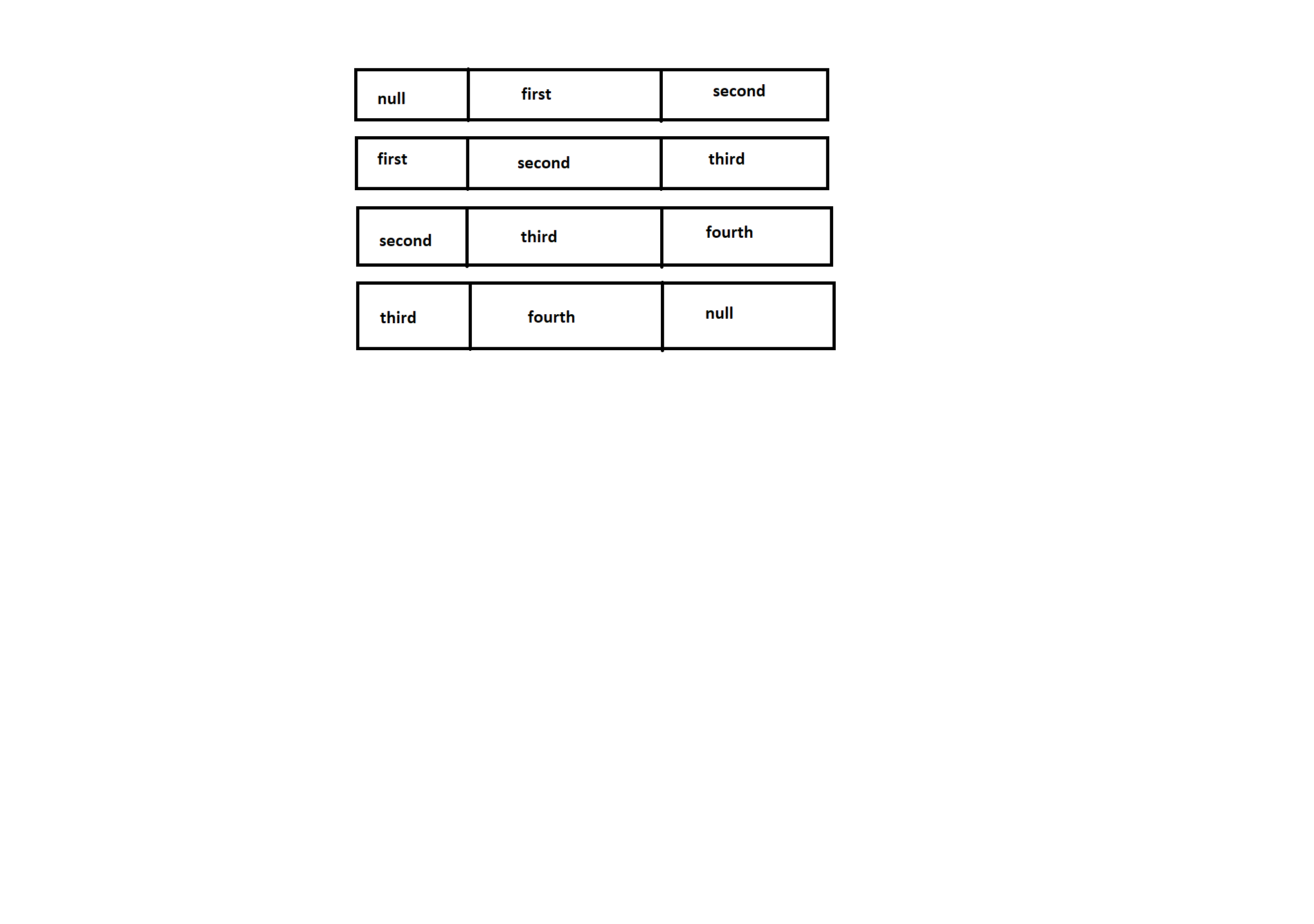
List.add(“first”);

List.add(“seconnd”);

List.add(“third”);

List.add(“fourth”);

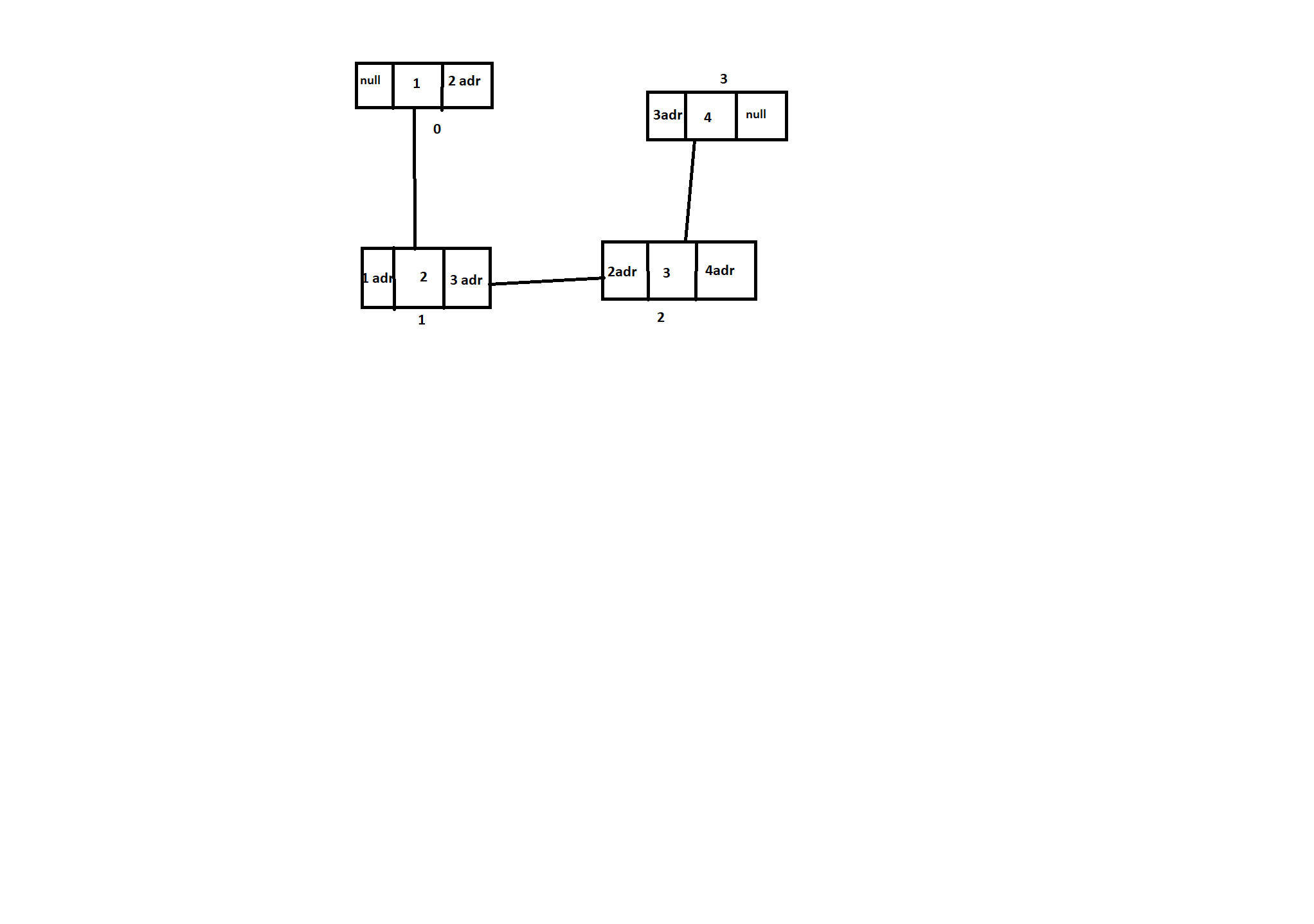
|  |  |  |
| --- | --- | --- |
| null | first | null |



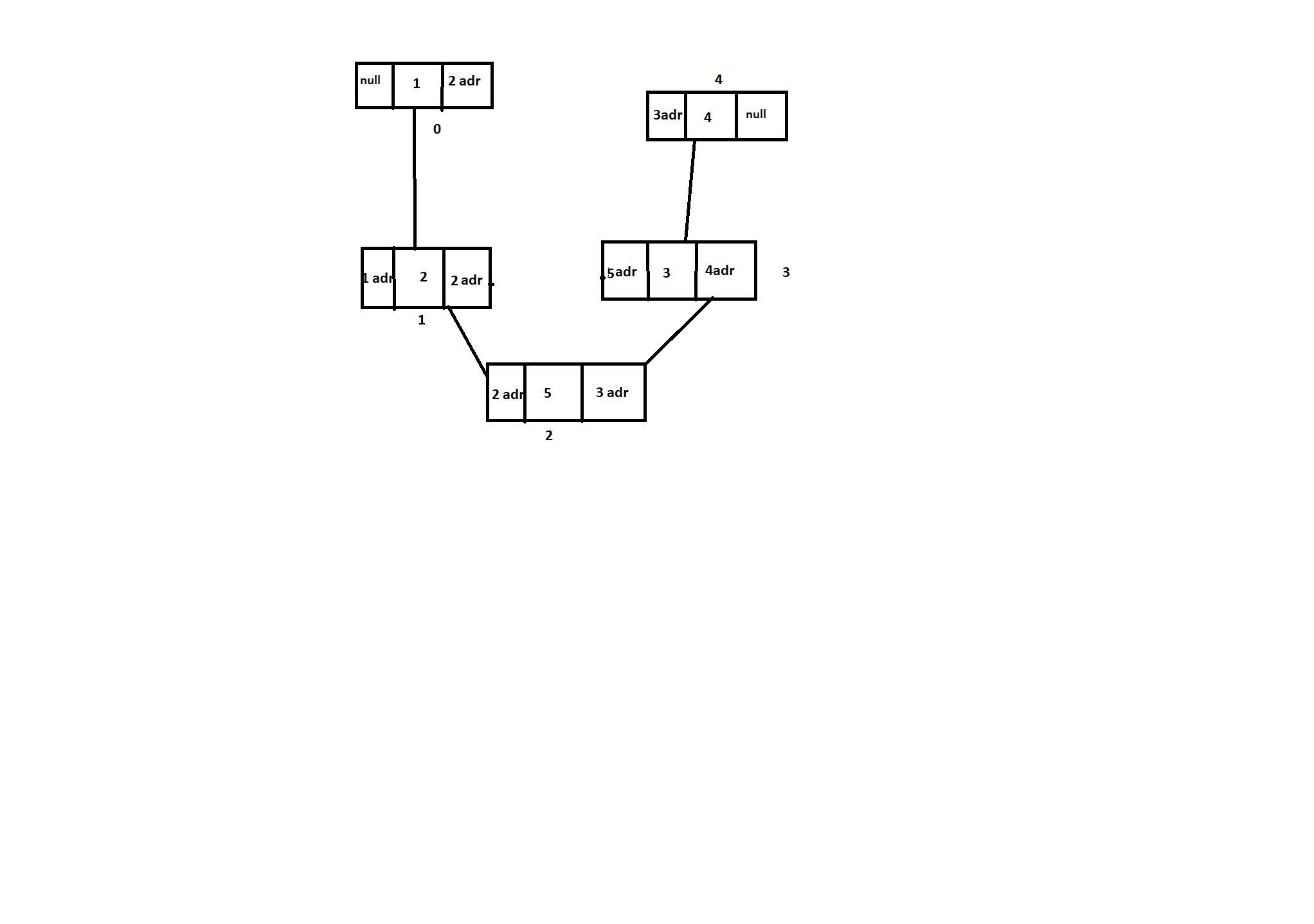
Nodes will have address and elements.

LinkedList is fast in insertion/update. It uses nodes it have previous and next element address. When we add an element at an index it will just add in that index an linked with the nodes between older nodes.

Ex:



add(2,5);



Here I’m adding an element at index 2 so it’s just created a new node and just updated the address of the previous node and next node, and also changes the index of the nodes, it will save time compare to ArrayList.

List 🡪

* + it follows insertion order.
  + It has listIterator
  + It is indexed
  + It allows duplication
  + We can use Collections.sort()

Sort🡪

* + it does not follow insertion order.
  + It does not have listIterator
  + It is not indexed.
  + It will not allow any duplication in order to achieve that we have to override the equals method and hashcode method.
  + It does not have Collections.sort()

HashMap🡪

* It contains key-pair values.
* It can not have duplicate keys.
* It may have one null key and multiple null values.
* It is non-synchronized.
* It does not provide a way to maintain the order of elements.
* The default capacity of the Java HashMap class is 16 with a load factor of 0.75.

LnkedHashMap🡪

* It stores the values in the key-pair combination.
* It can not have duplicate elements.
* It may have one null key and multiple null values.
* It is non synchronized.
* It provides an easy way to maintain the insertion order

**Differences Between HashSet, LinkedHashSet and TreeSet In Java**

**How they work internally?**

* HashSet uses HashMap internally to store it’s elements.
* LinkedHashSet uses LinkedHashMap internally to store it’s elements.
* TreeSet uses TreeMap internally to store it’s elements.

**Order Of Elements**

* HashSet doesn’t maintain any order of elements.
* LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are
* inserted.
* TreeSet orders the elements according to supplied Comparator. If no comparator is

supplied, elements will be placed in their natural ascending order.

**Performance**

* HashSet gives better performance than the LinkedHashSet and TreeSet.
* The performance of LinkedHashSet is between HashSet and TreeSet. It’s performance is

almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList

internally to maintain the insertion order of elements.

* TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the

elements after each insertion and removal operations.

**Insertion, Removal And Retrieval Operations**

* HashSet gives performance of order O(1) for insertion, removal and retrieval operations.
* LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval

operations.

* TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval

operations.

**How they compare the elements?**

* HashSet uses equals() and hashCode() methods to compare the elements and thus

removing the possible duplicate elements.

* LinkedHashSet also uses equals() and hashCode() methods to compare the elements.
* TreeSet uses compare() or compareTo() methods to compare the elements and thus

removing the possible duplicate elements. It doesn’t use equals() and hashCode() methods for comparision of elements.

**Null elements**

* HashSet allows maximum one null element.
* LinkedHashSet also allows maximum one null element.
* TreeSet doesn’t allow even a single null element. If you try to insert null element into

TreeSet, it throws NullPointerException.

**Memory Occupation**

* HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap

internally to store its elements.

* LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements.
* TreeSet also requires more memory than HashSet as it also maintains Comparator to sort

the elements along with the TreeMap.

**When To Use?**

* Use HashSet if you don’t want to maintain any order of elements.
* Use LinkedHashSet if you want to maintain insertion order of elements.
* Use TreeSet if you want to sort the elements according to some Comparator.

**Static block:**

Static block is used for the static initialization of a class. The code inside the block only executed for once when the first time the class is loaded into the memory.

We can print in console using static block without main method in the 1.6 previous versions. In these previous versions only without main method we execute and print the messages in the console using static block but 1.6 and after versions throws an error.

Ex: class Static{

Static(){

System.out.println("no arg consructor ");

}

static{

System.out.println("hello im from static block");

}

}

Error: Main method not found in class Static, please define the main method as:

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

**NOTE:** static block will be executed before the constructor.

**Servlets**

A servlet is a small java program which runs on the web server. Servlets receives request and send the response to the clients using HTTP protocols.

A servlet is an API. When a client sends a request through the browser the server receives that request and it will find the resource and send it as a response to the client.

When a client request to the server it create a thread for that request and it create an request object and response object after it will look for service method in the servlet.

Server

(servlet)

Thread

Request

Response

(bytes)

Client Request

Response

(bytes)

**Servlet Life Cycle:**

there are totally 5 methods in the servlet

1. init([**ServletConfig**](https://docs.oracle.com/javaee%2F7%2Fapi%2F%2F/javax/servlet/ServletConfig.html) config)
2. service(ServletRequest req, ServletResponse res)
3. getServletConfig()
4. getServletInfo()
5. destroy()

the life cycle of servlet dpends on three methods which are

* create object for servlet
* init(ServletConfig cofig)
* service(ServletRequest req,ServletResponse res)
* destroy()

**init(ServletConfig config)🡪** which is used to intialise the variables of servlet. Because in servlet the servlets are run in the server when we create a servlet the server will create an object for the servlet. To create an servlet object it look for the no argument constructor when it find the no argument constructor in servlet then only it creates an object. If we didn’t write an no argument constructor instead if we write parameter constructor it will not create an object. So no argument constructor is compulsory. If we wrote no argument constructor the we write parameter constructor then it will executed.

Server is creating an object for the servlet we can’t initialize or declare the variable so using init() method we can declare and initialize the variables in the servlet when the server encounters this init() method it will initializes the variables in the servlet.

**service(ServletRequest req,ServletResponse res)🡪** the service method is used to get the request from the client and send the response to the client. Before this the server will create an thread for each request made by client and after creating thread it will create request object and response object and it will look for the service method to processing the request and response.

**destroy()🡪** destroy method is used to clear the variables.

For an example if I create a connection to data base in the init() method in the destroy method I will close the connection.

EX for servlet:🡪 <https://github.com/deepthins08/Servlets>