C1 C2

Seats= 50;

50-10=40 seats are available

Theater theatre = new Theater ();

Seats= 50;

50-45=5 seat available

Theater theatre = new Theater ();

C1 C2

Seats= 50;

50-45=5 seats are available

Theater theatre = new Theater ();

Private constructor is used so that we can’t create object outside the class.

Static variable - It is static so every instance of the Singleton type will use the same variable, hence the "singleton" pattern.

Static means n number of objects will access the same memory.

**Try with resource:**

Complexity reduced

Length of the code decreased

We are not required to close the resource explicitly

**Polymorphism**

It helps the programmer to reuse the codes, We can write a method that can process different types of functionality with same name.

Exp

In this example, we are creating two classes Animal and Lion. Lion class extends Animal class and overrides its sound() method and eat(). We are calling the sound and eat method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

**RECURSION**

Calling a method by itself is known as recursion.

When there is a recursive call, a proper condition is given to return the call otherwise it should be stack overflow.

**MULTITHREADING**

Multiple threads executed simultaneously at a time is called multi-threading.

Each thread is an independent task of same program.

Every object having one unique lock.

Thread is a pre-defined class present in lang package.

**Advantages:**

We can perform many operations together, so it saves time.

Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

**Ways of creating thread**

By extending thread class

Exp:

By implementing runnable interface

Exp:

**Thread scheduler**

It is a part of jvm which is used to schedule the thread that which thread is execute first. But some OS system doesn’t support thread priority.

T1

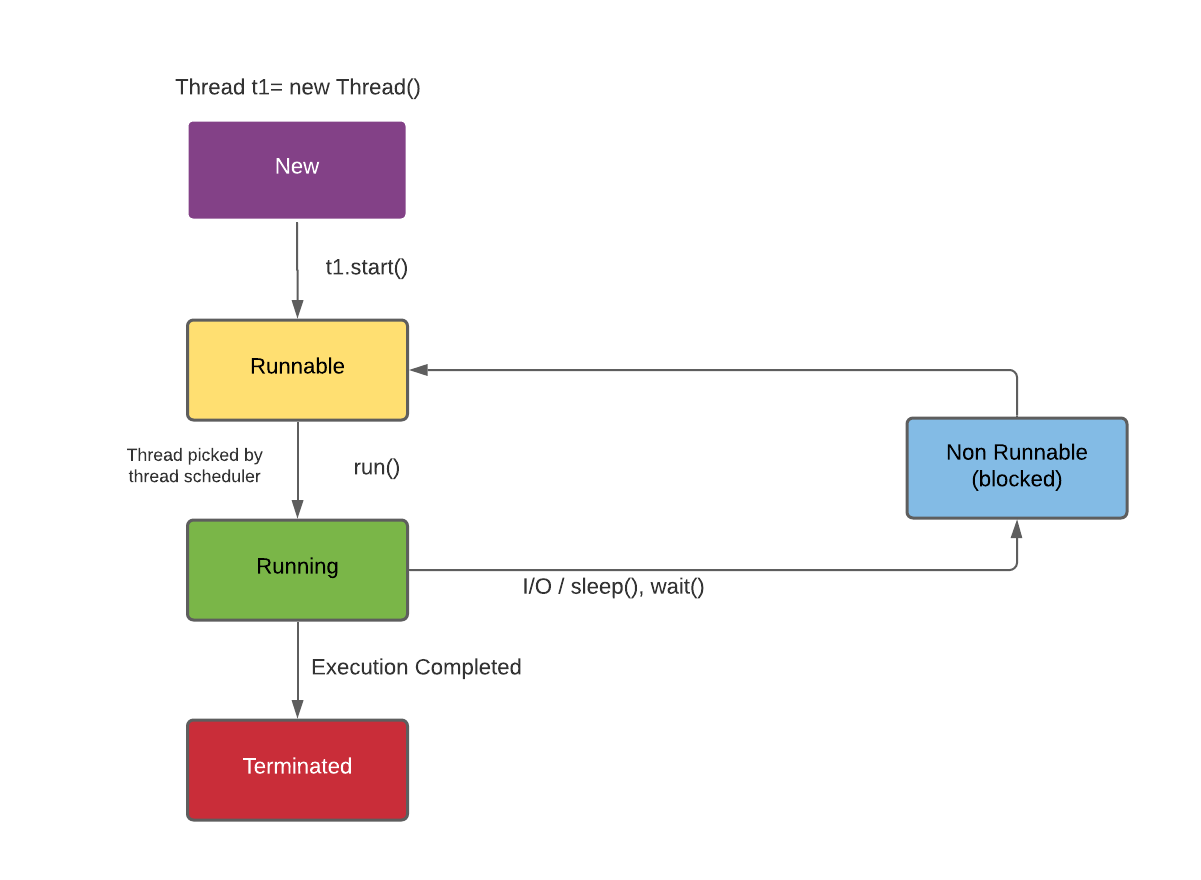
Thread sch**ed**uler

T2

CPU

T3

**Thread lifecycle**



**New:** Whenever a new thread is created by using new keyword, it is always in the new state.

**Runnable/Active:** When a thread invokes the start() method, it moves from the new state to the active state.

**Running:** When a thread picked by thread scheduler then thread is in running state.

**Terminated/dead:** When a thread executed completely by thread scheduler then thread is in dead state, if we want to put a thread from running stage to dead stage then we use stop().

**Waiting/blocked**: when a thread is in running state and we want to put it in waiting state then we can call sleep(), join().

**sleep()**

It is a static method in thread class that is used to put a thread into temporary waiting state. It gives interrupted exception that it put the thread from running state to waiting state.

**join()**

It is a method in thread class which allow a thread executed completely among all the thread present. It put a checked exception as interrupted exception as it put other thread from running state to waiting state.

**Synchronization**

It is a technique through which we can control multiple threads or we can say among all the thread present only one thread can enter into the synchronized area.

The main purpose of this technology is to overcome the problem of multithreading. When multiple threads are trying to access the same resources at same time, in that situation it may provide some wrong result.

Every object has a unique luck in java and this luck can be given to only one thread.

When we created synchronized method static then luck is not created with respect to object, it is created with respect to class.

**GENERIC**

Generic means parameterising datatype.

It is a non-primitive datatype provided by the user while using classes and interface.

It is declared inside angular bracket.

**Advantages:**

**Type-safety:** We can hold only a single type of objects in generics.

**Type casting is not required:** There is no need to typecast the object.

**Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime.

Exp:

**Type parameter:**

T – type

E – element

K – key

V – value

## **Generic class**

A class that can refer to any type is known as a generic class.

Exp:

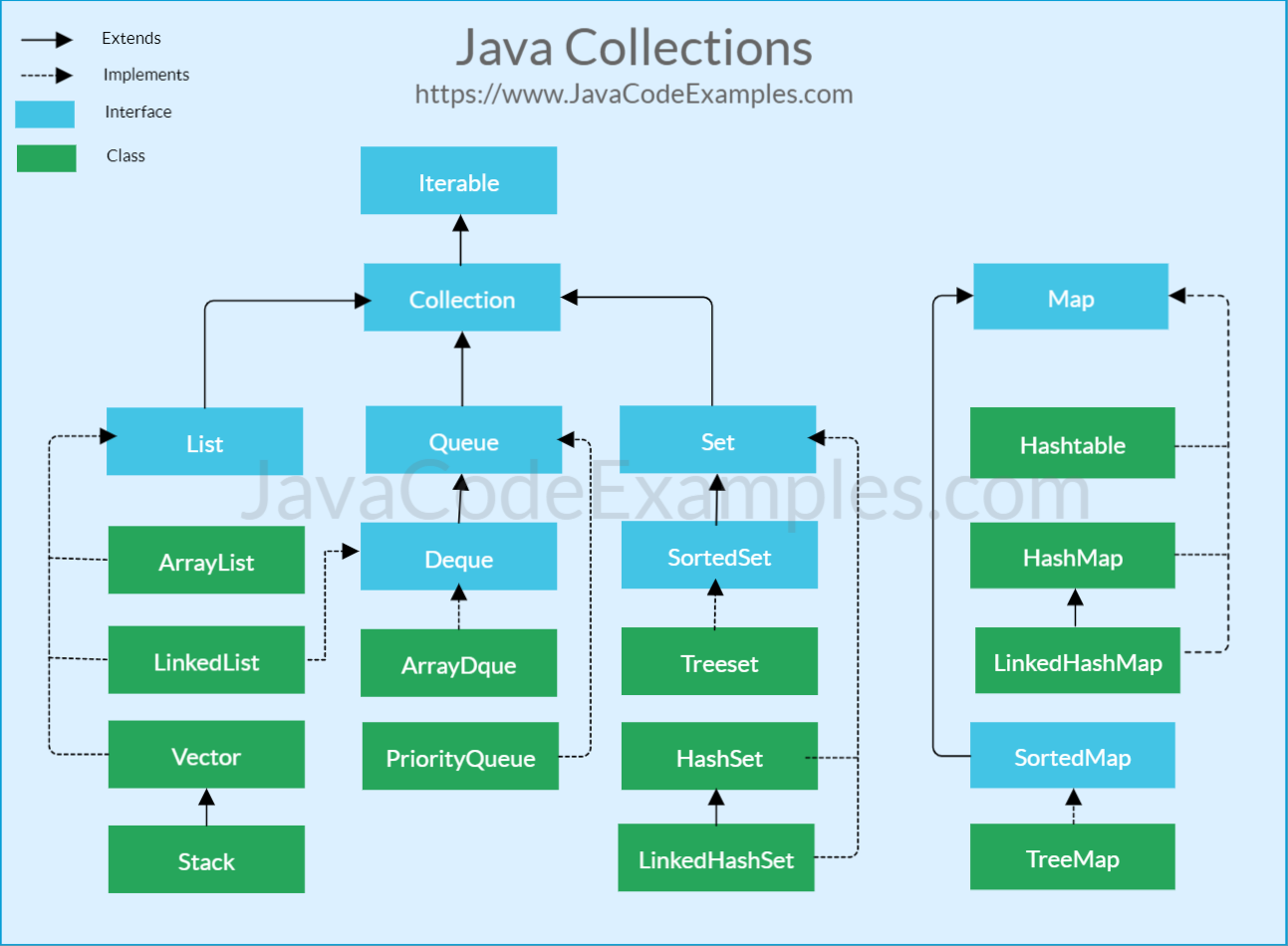
**Generic method:**

We can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared. It allows static as well as non-static methods.

Exp:

**COLLECTION**

Collection is a pre define class and interface that helps programmer to perform different types of data structure operation like sorting, storing, searching etc.



**Q. What is the disadvantages of array and when we go for collection?**

Array size is fixed in java, once it is created same array size can’t be increased or decreased. Collection is a pre-defined class and interface that is used to perform various types of data structure operation. In development whenever requirement we need to increase the size of array which is done by collection because it is not fixed in size, automatically it increases the size when reached at the maximum range. There is no default support from array to sorting, storing but collection have default method to doing all these things.

**Q. Similarities and distinguish between array and array list?**

**Similarities:**

Array and array list is used to store elements.

Both can store null values and duplicate values.

**Distinguish:**

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Size is fixed | Size is not fixed |
| Dynamically created | Class present in collection frame work |
| Provide length variable to find length of array | Provide size() to find size of array list |
| Homogeneous elements are present | Heterogeneous elements are present |
| Mandatory to provide size at the time of initialization | We can create an instance of ArrayList without specifying size. |
|  |  |

**Methods of Iterable interface**

It contains only one abstract method. i.e.,

Iterator<T> iterator()

It returns the iterator over the elements of type T.

**Methods of collection interface**

|  |  |
| --- | --- |
| Public boolean add(E e) | Insert element to the collection |
| Public boolean remove((Object element) | Delete an element from collection |
| Public int size() | Return total number of element present in collection |
| Public void clear() | Remove total number of elements from collection |
| Public boolean isEmpty() | Check if collection is empty |
| Public boolean contains(Object element) | Search an element |
| Public Iterator iterator() | Return an iterator |
| Public Object[] toArray() | Converts collection into array |

**List**

List is an index-based collection.

It allows duplicate objects.

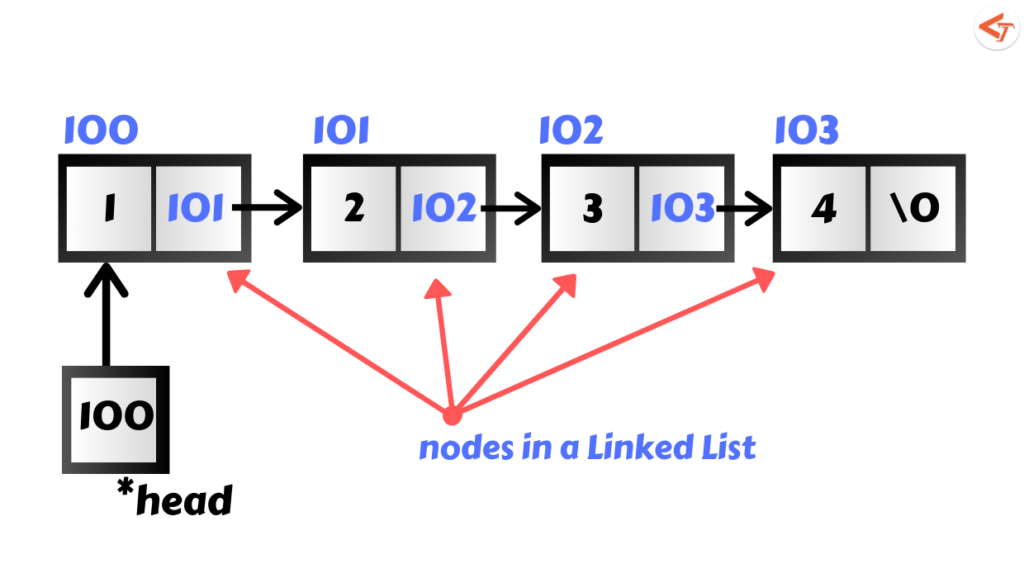
**ArrayList**

|  |  |
| --- | --- |
| add (E e) | Insert element in array |
| add (int index, E e) | Insert specific element in specific position |
| clear () | Remove all the element from this list |
| get (int index) | Fetch the element from particular position |
| isEmpty () | Return true if list is empty else return false |
| size () | Return number of elements present in list |
| remove () | Remove the elements from the specific position |
|  |  |

**LinkedList**

It connects with other node through next reference. In linked list data is stored in nodes and that have reference to the next node so adding element is simple as creating the node that updating the next pointer on the last node.

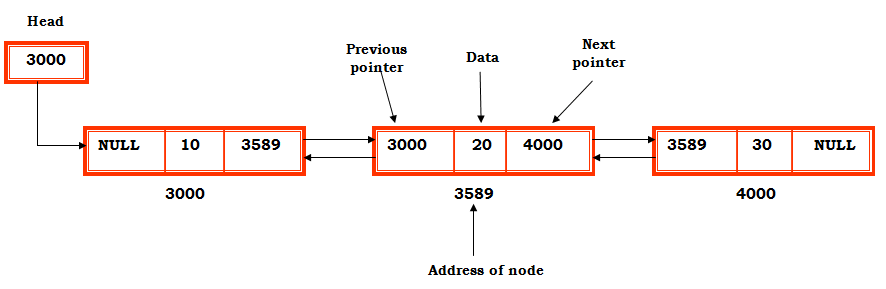
Deletion in linked list is fast because it involves only updating the next pointer in the node before deleting the node.



**DoubleLinkedList**

It connects with other node through next and previous reference. In double linked list data is stored in nodes and that have reference to the next and previous node so adding element is simple as creating the node that updating the next pointer on the last node.

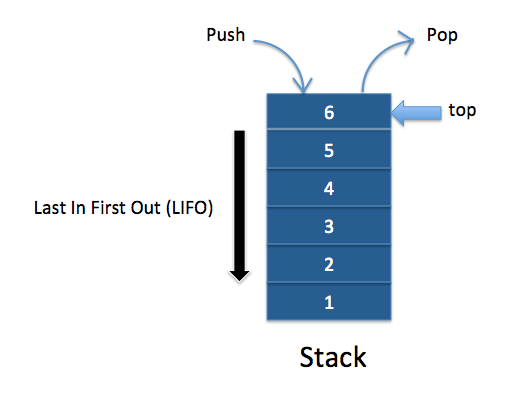
Deletion in double linked list is fast because it involves only updating the next pointer in the node before deleting the node. And update the previous pointer in the node after deleting the node.



**Stack**

It is the group of elements which is not index based.

It follows LIFO (Last in Fast Out).

It allows duplicate. Contain only default constructor.

Methods:

|  |  |
| --- | --- |
| push (E e) | Insert element in stack |
| pop () | The method removes an element from the top of the stack and returns the same element as the value of that function. |
| peek () | Looks at the top of the element of the stack without removing it |
| isEmpty () | Check that stack is empty or not |

**Queue**

Queue is the collection of elements which is not index based.

It allows duplicate.

It follows 1st in 1st out.

Linked list and array deque follows 1st in 1st out but priority queue sort the element according to comparable or comparator interface.

|  |  |
| --- | --- |
| add (E e) | Add the element to the queue |
| Peek() | Return the front element of the queue without removing it |
| E Poll() | Remove the head front element of the queue and return it, if queue is empty, it returns null |
| size() | Return no. of element present in queue |
|  |  |

**Set**

The **set** is an interface available in the **java.util** package.

The **set** interface extends the Collection interface.

Set is a collection of distinct elements.

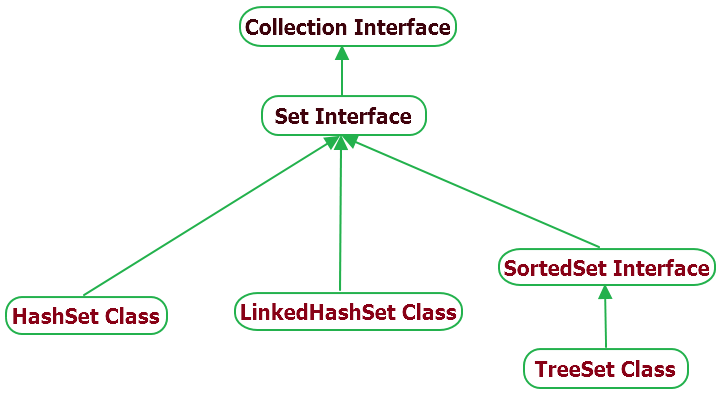
It can’ allow duplicate and we can’t add null values.

It is not an index-based collection.

HashSet iterate in random order.

Linked HashSet iterate in insertion order.

Tree Set iterate in sorting order.



Exp:

**Map**

Map is the collection of key and value pair. Key should not be duplicate.

For map we have 2 generic types. 1st indicates key and 2nd indicate s value.

Map<k,v>

When a new key and value is added, put() method will search for the current key, if the current key is already exist then value is updated and return the old value. If the given key is not exist then new key value is added and return null.

**Iterator interface**

Used to iterate element from Iterable type.

Object next()

boolean hasNext()

void remove()

**List Iterator Interface**

Used only for list type.

Object next()

Object previous()

boolean hasNext()

boolean hasPrevious()

int nextIndex()

int previousIndex()

void remove()