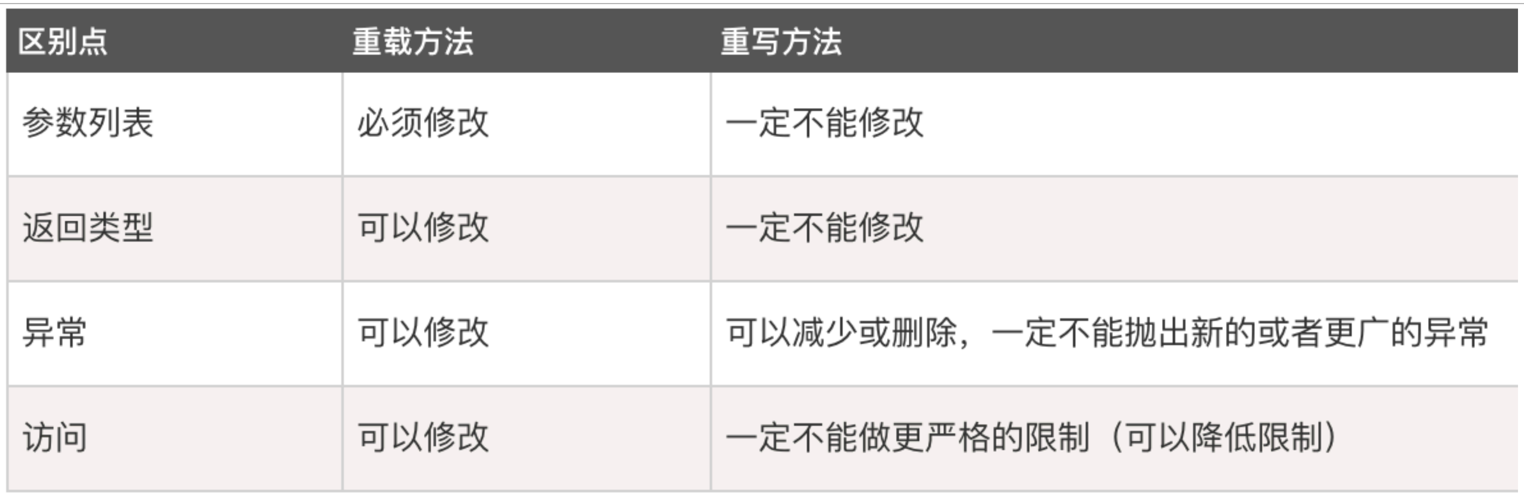
Overloading vs. Overriding

- Overloading is a term used to describe when two methods **in one class** have the **same name** but differ in the **type or number of arguments**
- Overriding occurs when a method shares **the same name and function signature** as another method in its super class. One of the methods is in the **parent class** and the other is in the **child class**. Overriding allows a child class to provide **a specific implementation** for a method that is already provided its parent class.

- 1). The real object type in the run-time, not the reference variable's type, determines which overridden method is used at runtime. In contrast, reference type determines which overloaded method will be used at compile time.
- 2). Polymorphism applies to overriding, not to overloading.
- 3). **Overriding is a run-time concept while overloading is a compile-time concept.**



区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改
异常	可以修改	可以减少或删除，一定不能抛出新的或者更广的异常
访问	可以修改	一定不能做更严格的限制（可以降低限制）

Linked List

public class **LinkedList<E>**
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable,
Like arrays, Linked List is a **linear data structure**. Unlike arrays, linked list elements are not stored at **contiguous location**; the elements are **linked using pointers**.
Representation:
A linked list is represented by a pointer to the **first node** of the linked list. The first node is called head. If the linked list is empty, then value of head is NULL.
Each node in a list consists of at least two parts:
1) data
2) Pointer (Or Reference) to the next node
In Java, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.
Advantages over arrays
1) Dynamic size
2) Ease of insertion/deletion
Drawbacks:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.
2) Extra memory space for a pointer is required with each element of the list.
3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.
链表是线性表的一种。线性表是最基本、最简单、也是最常用的一种数据结构。线性表中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其它数据元素都是首尾相接的。线性表有两种存储方式，一种是顺序存储结构，另一种是链式存储结构。我们常用的数组就是一种典型的顺序存储结构。
相反，链式存储结构就是两个相邻的元素在内存中可能不是相邻的，每一个元素都有一个指针域，指针域一般是存储着到下一个元素的指针。这种存储方式的**优点**是插入和删除的时间复杂度为 O(1)，不会浪费太多内存，添加元素的时候才会申请内存，删除元素会释放内存。缺点是访问的时间复杂度最坏为 O(n)。
顺序表的特性是随机读取，也就是访问一个元素的时间复杂度是O(1)，链式表的特性是插入和删除的时间复杂度为O(1)。
链表就是链式存储的线性表。根据指针域的不同，链表分为**单向链表、双向链表、循环链表**等等。

ArrayList

ArrayList is a part of collection framework and is present in java.util package. It provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

- ArrayList inherits AbstractList class and implements List interface.
- ArrayList is initialized by a size, however the size can increase if collection grows or shrunk if objects are removed from the collection.
- Java ArrayList allows us to randomly access the list.
- ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.
- ArrayList in Java can be seen as similar to vector in C++.

ArrayList Array

An array is basic functionality provided by Java. ArrayList is part of collection framework in Java. Therefore array members are accessed using [], while ArrayList has a set of methods to access elements and modify them.

Array is a fixed size data structure while ArrayList is not. One need not to mention the size of Arraylist while creating its object. Even if we specify some initial capacity, we can add more elements.

Array can contain both primitive data types as well as objects of a class depending on the definition of the array. However, ArrayList only supports object entries, not the primitive data types.

Since ArrayList can't be created for primitive data types, members of ArrayList are always references to objects at different memory locations (See this for details). Therefore in ArrayList, the actual objects are never stored at contiguous locations. References of the actual objects are stored at contiguous locations.
In array, it depends whether the arrays is of primitive type or object type. In case of primitive types, actual values are contiguous locations, but in case of objects, allocation is similar to ArrayList.

Stack	public class **Stack<E>** extends Vector<E> Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). **Implementation:** There are two ways to implement a stack: • Using array • Using linked list Java Collection framework provides a Stack class which models and implements Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.
Queue	public interface **Queue<E>** extends Collection<E> A Queue is a **linear structure** which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). The Queue interface is available in java.util package and extends the Collection interface. The queue collection is used to hold the elements about to be processed and provides various operations like the insertion, removal etc. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of list i.e. Being an interface the queue needs a concrete class for the declaration and the most common classes are the PriorityQueue and LinkedList in Java.It is to be noted that both the implementations are not thread safe. PriorityBlockingQueue is one alternative implementation if thread safe implementation is needed. Few important characteristics of Queue are: • The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept. • The Java Queue supports all methods of Collection interface including insertion, deletion etc. • LinkedList, ArrayBlockingQueue and PriorityQueue are the most frequently used implementations. • If any null operation is performed on BlockingQueues, NullPointerException is thrown.
Priority Queue	Priority Queue - 优先队列 应用程序常常需要处理带有优先级的业务，优先级最高的业务首先得到服务。因此优先队列这种数据结构应运而生。优先队列中的每个元素都有各自的优先级，优先级最高的元素最先得到服务；优先级相同的元素按照其在优先队列中的顺序得到服务。 优先队列可以使用数组或链表实现，从时间和空间复杂度来说，往往用二叉堆来实现。 Java 中提供`PriorityQueue`类，该类是 Interface Queue 的另外一种实现，和`LinkedList`的区别主要在于排序行为而不是性能，基于 priority heap 实现，非`synchronized`，故多线程下应使用`PriorityBlockingQueue`. 默认为自然序（小根堆），需要其他排序方式可自行实现`Comparator`接口，选用合适的构造器初始化。使用迭代器遍历时不保证有序，有序访问时需要使用`Arrays.sort(pq.toArray())`. 不同方法的时间复杂度： • enqueuing and dequeuing: `offer`, `poll`, `remove()` and `add` - O(logn)O(\log n)O(logn) • Object: remove(Object) and contains(Object) - O(n)O(n)O(n) • retrieval: `peek`, `element`, and `size` - O(1)O(1)O(1).
Binary Tree	A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child. 1. Data 2. Pointer to left child 3. Pointer to right child **Full Binary Tree** A Binary Tree is full if every node has 0 or 2 children. Following are examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaves have two children. **Complete Binary Tree:** A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible **Perfect Binary Tree** A Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at the same level. Balanced Binary Tree A binary tree is balanced if the height of the tree is O(Log n) where n is the number of nodes. For Example, AVL tree maintains O(Log n) height by making sure that the difference between heights of left and right subtrees is 1. Red-Black trees maintain O(Log n) height by making sure that the number of Black nodes on every root to leaf paths are same and there are no adjacent red nodes. Balanced Binary Search trees are performance wise good as they provide O(log n) time for search, insert and delete. **A degenerate (or pathological) tree** A Tree where every internal node has one child. Such trees are performance-wise same as linked list.
BST	**Binary Search Tree** is a node-based binary tree data structure which has the following properties: • The left subtree of a node contains only nodes with keys lesser than the node's key. • The right subtree of a node contains only nodes with keys greater than the node's key. • The left and right subtree each must also be a binary search tree.

Heap (Binary Heap)

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree

A Binary Heap is a Binary Tree with following properties.
1) It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.
2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.
一般情况下，堆通常指的是二叉堆，二叉堆是一个近似完全二叉树的数据结构，即披着二叉树羊皮的数组，故使用数组来实现较为便利。子结点的键值或索引总是小于（或者大于）它的父节点，且每个节点的左右子树又是一个二叉堆(大根堆或者小根堆)。根节点最大的堆叫做最大堆或大根堆，根节点最小的堆叫做最小堆或小根堆。常被用作实现优先队列。

1. 以数组表示，但是以完全二叉树的方式理解。
2. 唯一能够同时最优地利用空间和时间的方法
3. 在索引从0开始的数组中：

- 父节点 i 的左子节点在位置(2*i+1)
- 父节点 i 的右子节点在位置(2*i+2)
- 子节点 i 的父节点在位置floor((i-1)/2)

Hashing

Hashing is an important Data Structure which is designed to use a special function called the **Hash function** which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.
Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.

1. $s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)$
2. $(s.charAt(0)-'a') * 26^{n-1} + (s.charAt(1)-'a') * 26^{n-2} + ... + (s.charAt(n-1)-'a')$

HashMap

public class **HashMap<K,V>**
extends AbstractMap<K,V>
implements Map<K,V>, Cloneable, Serializable
Hash table based implementation of the `Map` interface. This implementation provides all of the optional map operations, and permits `null` values and the `null` key. (The `HashMap` class is roughly equivalent to `Hashtable`, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

- HashMap is a part of java.util package.
- HashMap extends an abstract class AbstractMap which also provides an incomplete implementation of Map interface.
- It also implements Cloneable and Serializable interface. K and V in the above definition represent Key and Value respectively.
- HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value.
- HashMap allows null key also but only once and multiple null values.
- This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time. It is roughly similar to HashTable but is unsynchronized.

Internally HashMap contains an array of Node and a node is represented as a class which contains 4 fields

1. int hash
2. K key
3. V value
4. Node next

TreeMap HashMap LinkedHashMap

- `HashMap` offers O(1) lookup and insertion. It is implemented by an array of linked lists.
- `TreeMap` offers O(log N) lookup and insertion. Keys are ordered. It is implemented by a Red-Black Tree.
- `LinkedHashMap` offers O(1) lookup and insertion. Keys are ordered by their insertion order. It is implemented by doubly-linked buckets.

	HashMap	TreeMap
Ordering	No	Yes
Implementation	Hashing	Red-Black Tree
Null Keys and Null values	One null key ,Any null values	Not permit null keys ,but any null values
Performance	O(1)	log(n)
Speed	Fast	Slow in comparison
Functionality	Provide Basic functions	Provide Rich functionality
Comparison	uses equals() method	uses compareTo() method
Interface implemented	Map	Navigable Map

HashTable	public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable 横向数组，纵向链表 search: 数值与Hashtable的长度做了取模的操作, 得到的数字直接作为hashtable中entry数组的index Synchronized not allow null keys and values
HashTable VS BST	一个设计良好(注意是设计良好的)的hash table 如下操作均为O(1) • Search • Insert • Delete 而self-balancing BST 这些操作均为O(logn) 所以在上面这些操作上hash table更优质, 但是如果有如下的需求场景, BST比hash table跟合适 • 得到所有的内容并且是sorted的 • order statistics, finding closest lower and greater elements, doing range queries are easy to do with BSTs • 对BST来说, 那些基本操作都是O(logn), 虽然hash table 的average time是O(1), 但某个特定操作为O(n) (resize hash table时)
解决hashtable冲突碰撞的方法	Separate chaining：即就是把相同的哈希值记录放在一个链表里面存储 **Advantages:** 1) Simple to implement. 2) Hash table never fills up, we can always add more elements to chain. 3) Less sensitive to the hash function or load factors. 4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted. **Disadvantages:** 1) Cache performance of chaining is not good as keys are stored using linked list. Open addressing provides better cache performance as everything is stored in same table. 2) Wastage of Space (Some Parts of hash table are never used) 3) If the chain becomes long, then search time can become O(n) in worst case. 4) Uses extra space for links. Open addressing：每个数组节点上就一个元素，如果待插入的Entry计算出来的hash值所在的数组节点非空闲（已经有一个Enry了），就采取某种方法再选择一个空闲的数组节点，插入该Enry。这种情况下，整个数组必须支持动态扩容：当数组空闲节点低于一个阈值时，将扩展数组容量为原来的一倍。这个阈值通常是0.72，也即数组节点有72%的比例为非空闲，就需要将数组扩容至原来的2倍
HashSet HashMap	HashSet class implements the Set interface HashMap class implements the Map interface In HashSet we store objects(elements or values) HashMap is used for storing key & value pairs. In short it maintains the mapping of key & value (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This is how you could represent HashMap elements if it has integer key and value of String type HashSet does not allow duplicate elements that means you can not store duplicate values in HashSet. HashMap does not allow duplicate keys however it allows to have duplicate values. HashSet permits to have a single null value. HashMap permits single null key and any number of null values.
Set	public interface **Set<E>** extends Collection<E> Set 是一种用于保存不重复元素的数据结构。常被用作测试归属性，故其查找的性能十分重要。 Set 与 Collection 具有安全一样的接口，通常有`HashSet`, `TreeSet` 或 `LinkedHashSet`三种实现。`HashSet`基于散列函数实现，无序，查询速度最快；`TreeSet`基于红-黑树实现，有序。
Graph	A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes. **1.** Adjacency Matrix **2.** Adjacency List
StringBuilder StringBuffer String	**String** is **immutable** (once created can not be changed)object . The object created as a String is stored in the Constant String Pool. Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously. String once assigned can not be changed. **StringBuffer** is **mutable** means one can change the value of the object . The object created through StringBuffer is stored in the **heap**. StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe . Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time . But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus StringBuilder is faster than the StringBuffer when calling the same methods of each class. String Buffer can be converted to the string by using toString() method. **StringBuilder** StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe. StringBuilder is fast as it is not thread safe .

Thread Process	**Process**: • An **executing instance** of a program is called a process. • Some operating systems use the term 'task' to refer to a program that is being executed. • A process is always stored in the **main memory** also termed as the primary memory or random access memory. • Therefore, a process is termed as an active entity. It disappears if the machine is rebooted. • **Several** process may be associated with a same program. • On a multiprocessor system, multiple processes can be executed in parallel. • On a uni-processor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency. Each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads. **Thread**: • A thread is a **subset** of the process. • It is termed as a 'lightweight process', since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel. • Usually, a process has only one thread of control – one set of machine instructions executing at a time. • A process may also be made up of multiple threads of execution that execute instructions concurrently. • Multiple threads of control can exploit the true parallelism possible on multiprocessor systems. • On a uni-processor system, a thread scheduling algorithm is applied and the processor is scheduled to run each thread one at a time. • All the threads running within a process share the same address space, file descriptors, stack and other process related attributes. • Since the threads of a process share the same memory, synchronizing the access to the shared data within the process gains unprecedented importance. A thread is an entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.
Java Javascript	Java is strongly typed language and variable must be declare first to use in program.In Java the type of a variable is checked at compile-time. JavaScript is weakly typed language and have more relaxed syntax and rules. Java is an object oriented programming language. JavaScript is an object based scripting language. Java applications can run in any virtual machine(JVM) or browser. JavaScript code run on browser only as JavaScript is developed for browser only. Objects of Java are class based even we can't make any program in java without creating a class. JavaScript Objects are prototype based. Java program has file extension ".Java" and translates source code into bytecodes which is executed by JVM(Java Virtual Machine). JavaScript file has file extension ".js" and it is interpreted but not compiled,every browser has the Javascript interpreter to execute JS code. Java is a Standalone laguage. contained within a web page and integrates with its HTML content. Java program uses more memory. JavaScript requires less memory therefore it is used in web pages. Java has a thread based approach to concurrency. Javascript has event based approach to concurrency.
Abstract Class	An abstract class is a class which **cannot be instantiated**. An abstract class is used by creating an inheriting subclass that can be instantiated. An abstract class does a few things for the inheriting subclass: 1. Define methods which can be used by the inheriting subclass. 2. Define abstract methods which the inheriting subclass must implement. 3. Provide a common interface which allows the subclass to be interchanged with all other subclasses. A Java class becomes abstract under the following conditions: 1. At least one of the methods is marked as abstract: 2. The class is marked as abstract: if a class has at least one pure virtual function, then the class becomes abstract an instance of an abstract class cannot be created, we can have references of abstract class type though. an abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of an inherited class is created. This allows us to create classes that cannot be instantiated, but can only be inherited. Abstract classes can also have final methods (methods that cannot be overridden). For example, the following program compiles and runs fine.

Interface Abstract Class	an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body). • Interfaces specify what a class must do and not how. It is the blueprint of the class. • An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement. • If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract. • A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection. 1. **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also. 2. **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables. 3. **Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables. 4. **Implementation:** Abstract class can provide the implementation of interface. Interface can't provide the implementation of abstract class. 5. **Inheritance vs Abstraction:** A Java interface can be implemented using keyword "implements" and abstract class can be extended using keyword "extends". 6. **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces. 7. **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.
Polymorphism 多态	多态是同一个行为具有多个不同表现形式或形态的能力, 同一个接口，使用不同的实例而执行不同操作 多态的优点 • 1. 消除类型之间的耦合关系 • 2. 可替换性 • 3. 可扩充性 • 4. 接口性 • 5. 灵活性 • 6. 简化性 多态存在的三个必要条件 • 继承(inherit) • 重写(override) • 父类引用指向子类对象(father -> child) Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways. Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. Like we specified in the previous chapter; **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways. 多态的实现方式 方式一：重写 方式二：接口 方式三：抽象类和抽象方法
Vector	public class **Vector<E>** extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable The Vector class implements a growable array of objects. Vectors basically falls in legacy classes but now it is fully compatible with collections. • Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index • They are very similar to ArrayList but Vector is synchronised and have some legacy method which collection framework does not contain. • It extends **AbstractList** and implements **List** interfaces. **Important points regarding Increment of vector capacity:** If increment is specified, Vector will expand according to it in each allocation cycle but if increment is not specified then vector's capacity get doubled in each allocation cycle. Vector defines three protected data member: • **int capacityIncreament:** Contains the increment value. • **int elementCount:** Number of elements currently in vector stored in it. • **Object elementData[]:** Array that holds the vector is stored in it.
Vector ArrayList	1. **Synchronization :** Vector is **synchronized**, which means only one thread at a time can access the code, while arrayList is **not synchronized**, which means multiple threads can work on arrayList at the same time. For example, if one thread is performing an add operation, then there can be another thread performing a remove operation in a multithreading environment. If multiple threads access arrayList concurrently, then we must synchronize the block of the code which modifies the list structurally, or alternatively allow simple element modifications. Structural modification means addition or deletion of element(s) from the list. Setting the value of an existing element is not a structural modification. 2. **Performance: ArrayList is faster**, since it is non-synchronized, while vector operations give slower performance since they are synchronized (thread-safe). If one thread works on a vector, it has acquired a lock on it, which forces any other thread wanting to work on it to have to wait until the lock is released. 3. **Data Growth:** ArrayList and Vector **both grow and shrink dynamically** to maintain optimal use of storage - but the way they resize is different. ArrayList increments 50% of the current array size if the number of elements exceeds its capacity, while vector increments 100% - essentially doubling the current array size. 4. **Traversal:** Vector can use both Enumeration and Iterator for traversing over elements of vector while ArrayList can only use **Iterator** for traversing.

Reflection	Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language. With that caveat in mind, reflection is a powerful technique and can enable applications to perform operations which would otherwise be impossible. 反射机制就是可以把一个类,类的成员(函数，属性)，当成一个对象来操作，希望读者能理解，也就是说，类，类的成员，我们在运行的时候还可以动态地去操作他们。 我们可以在**运行时**获得程序或程序集中每一个类型的成员和成员的信息。 Provides a way to get relfective information about Java classes and objects, and perform operations such as: • Getting information about the methods and fields present inside the class at runtime. • Creating a new instance of a class • Getting and setting the object fields directly by getting field reference, regardless of what the access modifier is. Three main reasons why Object Reflection is Useful: 1. It can help you observe or manipulate the runtime behavior of applications 2. It can help you debug or test programs, as you have direct access to methods, constructors, and fields. 3. You can call methods by name when you don't know the method in advance. Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime. • The required classes for reflection are provided under java.lang.reflect package. • Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object. • Through reflection we can invoke methods at runtime irrespective of the access specifier used with them. Reflection can be used to get information about - **1. Class** The getClass() method is used to get the name of the class to which an object belongs. **2. Constructors** The getConstructors() method is used to get the public constructors of the class to which an object belongs. **3. Methods** The getMethods() method is used to get the public methods of the class to which an objects belongs.
Garbage Collector	垃圾回收是Java用来将程序员从分配和释放内存的琐事中解放出来的自动过程 Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. In Java, process of deallocating memory is handled automatically by the garbage collector. The basic process can be described as follows. **Step 1: Marking** The first step in the process is called marking. This is where the garbage collector identifies which pieces of memory are in use and which are not **Step 2: Normal Deletion** Normal deletion removes unreferenced objects leaving referenced objects and pointers to free space **Step 2a: Deletion with Compacting** To further improve performance, in addition to deleting unreferenced objects, you can also compact the remaining referenced objects. By moving referenced object together, this makes new memory allocation much easier and faster. Serial Garbage Collector 串行垃圾回收器控制所有的应用线程。它是为单线程场景设计的，只使用一个线程来执行垃圾回收工作。它暂停所有应用线程来执行垃圾回收工作的方式不适用于服务器的应用环境。它最适用的是简单的命令行程序 **Parallel Garbage Collector** 并行垃圾回收器也称作基于吞吐量的回收器。它是JVM的默认垃圾回收器。与Serial不同的是，它使用多个线程来执行垃圾回收工作。和Serial回收器一样，它在执行垃圾回收工作是也需要暂停所有应用线程。 **CMS Garbage Collector** 并发标记清除(Concurrent Mark Sweep,CMS)垃圾回收器，使用多个线程来扫描堆内存并标记可被清除的对象，然后清除标记的对象。CMS垃圾回收器只在下面这两种情形下暂停工作线程， **G1 Garbage Collector** G1垃圾回收器应用于大的堆内存空间。它将堆内存空间划分为不同的区域，对各个区域并行地做回收工作。G1在回收内存空间后还立即堆空闲空间做整合工作以减少碎片。CMS却是在全部停止(stop the world,STW)时执行内存整合工作。对于不同的区域G1根据垃圾的数量决定优先级。
Keywords	• `private` should be used when something is not used outside of a given class ◦ for methods and fields - when they are used only within the same class ◦ for classes - only on nested classes, when used in the same class • `protected` should be used when ◦ for methods and field - when you need to make them accessible to subclasses only ◦ for classes - again only nested classes, accessible by subcalsses • `public` is used when something is accessible by every other class • `static` is used when you don't need an instance of a class (i.e. object) to use it: ◦ for fields - when you want to have a global field ◦ for methods - when you need utility functions that do not depend on object state ◦ for nested classes - when you want to access them without an instance of the enclosing class. • `abstract` when you don't want to provide implementations in the current class: ◦ on methods - when subclasses have to provide the actual implementation, but you want to invoke these methods (no matter how they are implemented) in this class. ◦ on classes - to denote that the class may have abstract methods. • `final` - when you don't want something to change. ◦ on fields, when you want to assign the value only once. It is useful when you want to pass a local variable to an inner class - you have to declare it final. ◦ on classes and methods - when you don't want subclasses to be able to extend / override them.

List
Set
Map

```
Collection
├List
│ ├LinkedList
│ ├ArrayList
│ └Vector
│   └Stack
└Set
Map
├Hashtable
├HashMap
└WeakHashMap
```

List：1.可以允许重复的对象。
2.可以插入多个null元素。
3.是一个有序容器，保持了每个元素的插入顺序，输出的顺序就是插入的顺序。
4.常用的实现类有 ArrayList、LinkedList 和 Vector。ArrayList 最为流行，它提供了使用索引的随意访问，而 LinkedList 则对于经常需要从 List 中添加或删除元素的场合更为合适。

Set：1.不允许重复对象
2. 无序容器，你无法保证每个元素的存储顺序，TreeSet通过 Comparator 或者 Comparable 维护了一个排序顺序。
3. 只允许一个 null 元素
4.Set 接口最流行的几个实现类是 HashSet、LinkedHashSet 以及 TreeSet。最流行的是基于 HashMap 实现的 HashSet；TreeSet 还实现了 SortedSet 接口，因此 TreeSet 是一个根据其 compare() 和 compareTo() 的定义进行排序的有序容器。

Map: 1.Map不是collection的子接口或者实现类。Map是一个接口。
2.Map 的 每个 Entry 都持有两个对象，也就是一个键一个值，Map 可能会持有相同的值对象但键对象必须是唯一的。
3. TreeMap 也通过 Comparator 或者 Comparable 维护了一个排序顺序。
4. Map 里你可以拥有随意个 null 值但最多只能有一个 null 键。
5.Map 接口最流行的几个实现类是 HashMap、LinkedHashMap、Hashtable 和 TreeMap。（HashMap、TreeMap最常用）

1. 如果你经常会使用索引来对容器中的元素进行访问，那么 List 是你的正确的选择。如果你已经知道索引了的话，那么 List 的实现类比如 ArrayList 可以提供更快速的访问,如果经常添加删除元素的，那么肯定要选择LinkedList。
2. 如果你想容器中的元素能够按照它们插入的次序进行有序存储，那么还是 List，因为 List 是一个有序容器，它按照插入顺序进行存储。
3. 如果你想保证插入元素的唯一性，也就是你不想有重复值的出现，那么可以选择一个 Set 的实现类，比如 HashSet、LinkedHashSet 或者 TreeSet。所有 Set 的实现类都遵循了统一约束比如唯一性，而且还提供了额外的特性比如 TreeSet 还是一个 SortedSet，所有存储于 TreeSet 中的元素可以使用 Java 里的 Comparator 或者 Comparable 进行排序。LinkedHashSet 也按照元素的插入顺序对它们进行存储
4. 如果你以键和值的形式进行数据存储那么 Map 是你正确的选择。你可以根据你的后续需要从 Hashtable、HashMap、TreeMap 中进行选择。

List接口有三个实现类：
1.1 LinkedList
基于链表实现，链表内存是散列的，增删快，查找慢；
1.2 ArrayList
基于数组实现，非线程安全，效率高，增删慢，查找快；
1.3 Vector
基于数组实现，线程安全，效率低，增删慢，查找慢；

Map接口有四个实现类：
2.1 HashMap
基于 hash 表的 Map 接口实现，非线程安全，高效，支持 null 值和 null
键；
2.2 HashTable
线程安全，低效，不支持 null 值和 null 键；
2.3 LinkedHashMap
是 HashMap 的一个子类，保存了记录的插入顺序；
2.4 SortMap 接口
TreeMap，能够把它保存的记录根据键排序，默认是键值的升序排序

Set接口有两个实现类：
3.1 HashSet
底层是由 Hash Map 实现，不允许集合中有重复的值，使用该方式时需要重写 equals()和 hash Code()方法；
3.2 LinkedHashSet
继承于 HashSet，同时又基于 LinkedHashMap 来进行实现，底层使用的是 LinkedHashMap

Object-oriented languages Principles	Object-oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. Object-oriented design is concerned with developing an object-oriented model of a software system to implement the identified requirements. Object Oriented Design can yield the following benefits: maintainability through simplified mapping to the problem domain, which provides for less analysis effort, less complexity in system design, and easier verification by the user; reusability of the design artifacts, which saves time and costs; and productivity gains through direct mapping to features of Object-Oriented Programming Languages. Object-oriented languages (OOLs) fall into two broad categories: • Class-based OOLs such as Smalltalk, C++, Java, Ruby, Scala, and Eiffel.In a class-based OOL objects are instances of and created by classes. • Classless OOLs such as Self and Haskel.In a classless OOL objects serve as prototypes for other objects; objects are cloned from their prototypes. Like many distinctions there is no true dichotomy here - the two categories are extremes of a spectrum. The JavaScript programming language, for example, does not fit cleanly into either category. It lies somewhere in between. Encapsulation封装 封装是面向对象的特征之一，是对象和类概念的主要特性。封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。 Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit, e.g., a class in Java. This concept is often used to hide the internal state representation of an object from the outside. There are two important aspects of encapsulation: • Access restriction - preventing one object from accessing another's internal state, for example. • Namespaces/scopes - allowing the same name to have different meanings in different contexts. Encapsulation mechanisms are essential for reducing couplings between software components. Many encapsulation mechanisms originated with non-object-oriented languages. Object-oriented languages add additional encapsulation mechanisms. Inheritance继承 面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展 Inheritance is the ability of one class to inherit capabilities or properties of another class, called the parent class. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class which possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy. There are two types of inheritance in OOLs • interface inheritance and • implementation inheritance. Interface inheritance is only necessary in typed OOLs. This is best understood when considering delegation-based design patterns. Implementation inheritance mechanisms depend on the type of OOL. • For class-based OOLs, classes inherit from classes. • For classless OOLs, objects inherit from objects. Polymorphism多态 多态性（polymorphisn）是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。 Polymorphism is the ability for data to be processed in more than one form. It allows the performance of the same task in various ways. It consists of method overloading and method overriding, i.e., writing the method once and performing a number of tasks using the same method name.
Interface	接口与类相似点： • 一个接口可以有多个方法。 • 接口文件保存在 .java 结尾的文件中，文件名使用接口名。 • 接口的字节码文件保存在 .class 结尾的文件中。 • 接口相应的字节码文件必须在与包名称相匹配的目录结构中。 接口与类的区别： • 接口不能用于实例化对象。 • 接口没有构造方法。 • 接口中所有的方法必须是抽象方法。 • 接口不能包含成员变量，除了 static 和 final 变量。 • 接口不是被类继承了，而是要被类实现。 • 接口支持多继承。 接口特性 • 接口中每一个方法也是隐式抽象的,接口中的方法会被隐式的指定为 **public abstract**（只能是 public abstract，其他修饰符都会报错）。 • 接口中可以含有变量，但是接口中的变量会被隐式的指定为 **public static final** 变量（并且只能是 public，用 private 修饰会报编译错误）。 • 接口中的方法是不能在接口中实现的，只能由实现接口的类来实现接口中的方法。 抽象类和接口的区别 • 1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。 • 2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 **public static final** 类型的。 • 3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。 • 4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。
Heap **Stack**	Stack Memory in Java is used for static memory allocation and the execution of a thread. Heap space in Java is used for dynamic memory allocation for Java objects and JRE classes at the runtime. 1. Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution. 2. Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space. 3. Objects stored in the heap are globally accessible whereas stack memory can't be accessed by other threads. 4. Memory management in stack is done in LIFO manner whereas it's more complex in Heap memory because it's used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at Java Garbage Collection. 5. Stack memory is short-lived whereas heap memory lives from the start till the end of application execution. 6. We can use **-Xms** and **-Xmx** JVM option to define the startup size and maximum size of heap memory. We can use **-Xss** to define the stack memory size. 7. When stack memory is full, Java runtime throws `java.lang.StackOverFlowError` whereas if heap memory is full, it throws `java.lang.OutOfMemoryError: Java Heap Space` error. 8. Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

Bit **Byte**	Bit: Binary digit（二进制数位）的缩写,意为“位”或“比特”，是计算机运算的基础, 计算机中的最小存储单元, 存储内容总是0或1，所有二进制状态的实体都可以使用1bit表示, 8bits组成1byte, 不能够单独寻址 The **byte** is one of the **primitive** data types in Java. This means that the data type comes packaged with the programming language and there is nothing special you have to do to get it to work. A Java byte is 8 bits, and can hold values ranging from -128 to 127, or 0 to 255.
List	The Java.util.List is a child interface of Collection. It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements. List Interface is implemented by ArrayList, LinkedList, Vector and Stack classes.
Two's **One's** **Complement**	Two's complement: one's complement + add 1 First we write out 28 in binary form. Then we invert the digits. 0 becomes 1, 1 becomes 0. Then we add 1. That is how one would write -28 in 8 bit binary. One's complement: Invert all the digits
equals **==**	equals: evaluates to the comparison of values in the objects ==: is a reference comparison, i.e. both objects point to the same memory location