**ArrayList:**

add(E element): O(1) amortized, O(n) in worst case

remove(int index): O(n)

get(int index): O(1)

contains(Object obj): O(n)

size(): O(1) - 返回ArrayList中元素的个数

isEmpty(): O(1) - 检查ArrayList是否为空

set(int index, E element): O(1) - 替换指定索引位置的元素

indexOf(Object obj): O(n) - 返回指定元素第一次出现的索引位置

lastIndexOf(Object obj): O(n) - 返回指定元素最后一次出现的索引位置

clear(): O(1) - 清空ArrayList中的所有元素

toArray(): O(n) - 将ArrayList转换为数组

**LinkedList:**

add(E element): O(1)

remove(int index): O(n)

get(int index): O(n)

contains(Object obj): O(n)

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

**Vector:**

add(E element): O(1) amortized, O(n) in worst case

remove(int index): O(n)

get(int index): O(1)

contains(Object obj): O(n)

**HashSet:**

add(E element): O(1) average case, O(n) worst case

remove(Object obj): O(1) average case, O(n) worst case

contains(Object obj): O(1) average case, O(n) worst case

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

**TreeSet:**

add(E element): O(log n)

remove(Object obj): O(log n)

contains(Object obj): O(log n)

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

**HashMap:**

put(K key, V value): O(1) average case, O(n) worst case

remove(Object key): O(1) average case, O(n) worst case

get(Object key): O(1) average case, O(n) worst case

containsKey(Object key): O(1) average case, O(n) worst case

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

**TreeMap:**

put(K key, V value): O(log n)

remove(Object key): O(log n)

get(Object key): O(log n)

containsKey(Object key): O(log n)

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

**Stack:**

push(E element): O(1)

pop(): O(1)

peek(): O(1)

isEmpty(): O(1)

size(): O(1)

**Queue:**

enqueue(E element): O(1)

dequeue(): O(1)

peek(): O(1)

isEmpty(): O(1)

size(): O(1)

**PriorityQueue:**

insert(E element): O(log n)

remove(): O(log n)

peek(): O(1)

isEmpty(): O(1)

**Binary Search Tree (BST):**

insert(E element): O(log n) average case, O(n) worst case (unbalanced tree)

remove(E element): O(log n) average case, O(n) worst case (unbalanced tree)

search(E element): O(log n) average case, O(n) worst case (unbalanced tree)

**Graph (Adjacency List representation):**

addVertex(V vertex): O(1)

removeVertex(V vertex): O(|V| + |E|)

addEdge(V source, V destination): O(1)

removeEdge(V source, V destination): O(|V|)

getNeighbors(V vertex): O(1)

**Trie (Prefix Tree):**

insert(String word): O(k), k is the length of the word

search(String word): O(k), k is the length of the word

startsWith(String prefix): O(k), k is the length of the prefix

**VL Tree（平衡二叉搜索树）：**

insert(E element): O(log n)

remove(E element): O(log n)

search(E element): O(log n)

**Red-Black Tree（红黑树）：**

insert(E element): O(log n)

remove(E element): O(log n)

search(E element): O(log n)

**Hash Table（哈希表）：**

insert(E element): O(1) average case, O(n) worst case

remove(E element): O(1) average case, O(n) worst case

search(E element): O(1) average case, O(n) worst case

put(K key, V value): O(1) amortized, O(n) in worst case

get(K key): O(1)

remove(K key): O(1)

containsKey(K key): O(1)

size(): O(1)

isEmpty(): O(1)

clear(): O(1)

LinkedHashSet:

add(E element): O(1)

remove(E element): O(1)

contains(Object obj): O(1)

size(): O(1)z

isEmpty(): O(1)

clear(): O(1)

在图论中，"trail"**、"circuit"和"pat**h"是三个不同的概念：

Trail（路径）：Trail是指图中的一条边序列，其中每条边都连接图中的相邻节点，且不重复经过同一节点。也就是说，Trail是一个不重复经过节点的边的序列。

Circuit（回路）：Circuit是指一条Trail，它的起点和终点是同一个节点。也就是说，Circuit是一个形成环路的Trail。

Path（路径）：Path是指图中的一条边序列，其中每条边都连接图中的相邻节点，且不重复经过同一节点。与Trail不同的是，Path的起点和终点是不同的节点。

简而言之，Trail是一条连接相邻节点的边的序列，Path是一条连接两个不同节点的边的序列，而Circuit是一条形成环路的Trail。

类（class）可以实现（implement）接口（interface），而类可以扩展（extend）其他类。

当一个类实现一个接口时，它必须提供接口中定义的所有方法的具体实现。通过实现接口，类可以获得接口定义的方法，并为其提供具体的功能实现。一个类可以同时实现多个接口。

当一个类扩展另一个类时，它继承了被扩展类的属性和方法，并且可以添加自己的属性和方法。类之间的继承关系可以形成类的层次结构，并且子类可以通过扩展父类来继承和重用代码。

所以，类可以实现接口和扩展其他类，这两种方式在Java中都是有效的

在Java中，以下内容不能被实例化：

抽象类（Abstract class）：抽象类是一个不完整的类，它只能作为其他类的基类，不能直接实例化。抽象类中可以包含抽象方法和具体方法，但是抽象方法必须由子类进行实现。

接口（Interface）：接口定义了一组方法的契约，用于规范类的行为。接口本身不能被实例化，但是类可以实现接口来遵循接口的契约。

枚举类型（Enum）：枚举类型是一种特殊的类，用于表示一组固定的常量值。枚举类型在定义时列出了所有可能的实例，因此无法在运行时实例化其他值。

抽象方法（Abstract method）：抽象方法是一种在抽象类或接口中声明但没有具体实现的方法。抽象方法不能被直接调用，只能在子类中进行实现后才能使用。

当涉及到类和接口的关系时，以下是一些基本概念的解释：

类（Class）：类是面向对象编程中最基本的概念之一。它是用于创建对象的蓝图或模板，描述了对象的属性（字段）和行为（方法）。类可以具有构造函数、实例变量、实例方法等。

接口（Interface）：接口定义了一组方法的规范，没有实现的具体代码。它定义了一种协议或契约，规定了实现该接口的类必须提供的方法。接口可以看作是一种约束，使得实现了接口的类具有相同的行为。接口中的方法默认是抽象的，不包含具体实现。

继承（Inheritance）：继承是一种类之间的关系，其中一个类（子类）可以继承另一个类（父类）的属性和方法。子类可以使用继承来扩展或修改父类的功能。通过继承，子类可以访问父类的公共（非私有）成员变量和方法。

实现（Implementation）：实现是指一个类实现了一个接口，并提供了接口中定义的方法的具体实现。一个类可以实现一个或多个接口。通过实现接口，类必须提供接口中定义的所有方法的实现。

基本的关系是：

类通过继承来扩展或修改其他类的功能。

类通过实现接口来定义一组规范，实现相同的行为。

使用关键字：

使用 extends 关键字表示一个类扩展（继承）另一个类。

使用 implements 关键字表示一个类实现一个或多个接口。

总结：

类可以继承另一个类的属性和方法，用于实现类之间的层次结构和代码重用。

接口定义了一组方法的规范，类可以实现接口以提供具体的方法实现。

类可以同时继承另一个类并实现一个或多个接口。

请注意，一个类只能继承一个父类，但可以实现多个接口。这种灵活性使得Java能够实现单继承和多重接口继承的特性。

**抽象类可以取实现接口？接口只能继承接口，不能继承类？**

总结：

抽象类可以同时实现接口和定义自己的抽象方法，可以提供部分实现。

接口只能继承其他接口，一个类可以实现多个接口。

类可以继承一个父类，但只能实现一个类，但可以实现多个接口。