Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет**

**имени первого Президента России Б.Н. Ельцина»**

**Институт радиоэлектроники и информационных технологий**

**Лабораторная работа №9**

«Введение в алгоритмы и структуры данных. Исследование возможностей коллекций в языке программирования Java»

Выполнил: студент РИЗ-130916y

Кузнецова Екатерина Алексеевна

Проверил: преподаватель

Архипов Николай Александрович

Екатеринбург

2024

**Цель:** приобретение навыков работы с рекурсивными методами, знакомство с динамическими структурами данных, приобретение навыков создания и использования простейшей динамической структуры, исследование возможностей коллекций в языке программирования Java, сравнение коллекций по времени выполнения операций, знакомство с реализацией коллекций

# 3 ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

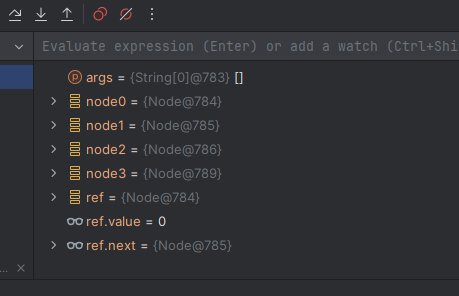
1. Создать приложения для демонстрации примеров 1 – 5 из раздела 1. Для примера 5 дополнительно вывести в консоль последовательность обхода дерева рекурсивных вызовов.  
public class nomer1 { public static int Fib(int n)  
        { if ((n == 0) || (n == 1))  
                return n; return Fib(n - 1) + Fib(n - 2);  
        } public static void main(String[] args){  
            System.out.println(Fib(0)); System.out.println(Fib(1));  
            System.out.println(Fib(2)); System.out.println(Fib(3));  
            System.out.println(Fib(4)); System.out.println(Fib(5));  
    }}

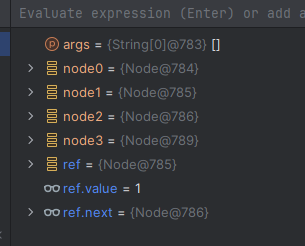
2. Создать приложение с использованием рекурсии для перевода целого числа, введенного с клавиатуры, в двоичную систему счисления.  
import java.util.Scanner;  
public class nomer1 { public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in); System.out.println("Введите целое число:");  
        int number = scanner.nextInt(); System.out.println("Двоичная запись числа: " + convertToBinary(number));  
    }  
    public static String convertToBinary(int number) { if (number == 0) {  
            return "0"; }  
        String remainder = String.valueOf(number % 2); return convertToBinary(number / 2) + remainder;  
    }}

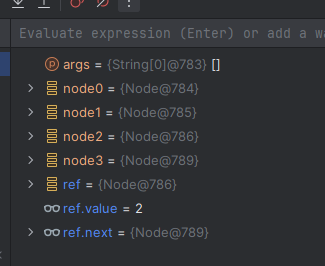
3. Создать приложение, позволяющее ввести и вывести одномерный массив целых чисел. Для ввода и вывода массива разработать рекурсивные методы вместо циклов for.

import java.util.\*;  
  
public class nomer1 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int[] array = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();  
 ar(array);  
 }  
 public static void ar(int[] array){  
 if (array.length == 0) {  
 return;  
 }  
 System.out.println(array[0]);  
 ar(Arrays.copyOfRange(array, 1, array.length));  
 }  
}

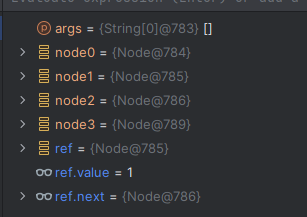
4. Выполнить пример 1 из раздела 2. Отработать код с помощью отладчика и привести скриншоты минимум трех точек, обработанных отладчиком.

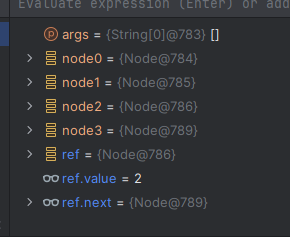


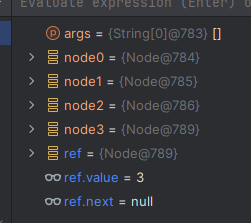


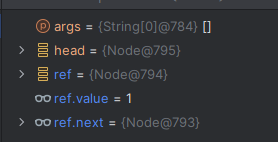


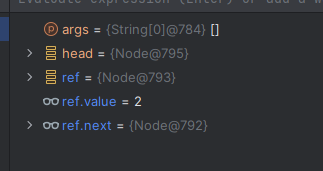
5. Создать два проекта, в которых продемонстрировать два способа создания линейного однонаправленного списка (с головы и с хвоста) согласно примеру 2 из второго раздела. Отработать код с помощью отладчика и привести скриншоты минимум трех точек, обработанных отладчиком.

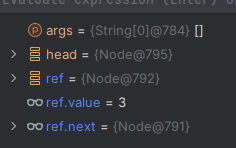












6. Заполнить HashMap 10 объектами <Integer, String>. Найти строки у которых *ключ >* 5. Если *ключ* = 0, вывести строки через запятую. Перемножить все ключи, где длина строки>5.

import java.util.HashMap;  
import java.util.Map;  
  
public class nomer1 {  
 public static void main(String[] args) {  
 Map<Integer, String> map = new HashMap<>();  
 for (int i = 0; i < 10; i++) {  
 map.put(i, "строка " + i);  
 }  
 for (Map.Entry<Integer, String> entry : map.entrySet()) {  
 if (entry.getKey() > 5) {  
 System.out.println(entry.getValue());  
 }  
 }  
  
 if (map.containsKey(0)) {  
 System.out.println(map.get(0));  
 }  
 int product = 1;  
 for (Map.Entry<Integer, String> entry : map.entrySet()) {  
 if (entry.getValue().length() > 5) {  
 product \*= entry.getKey();  
 }  
 }  
 System.out.println("Произведение ключей, где длина строки > 5: " + product);  
 }  
}

7. В кругу стоят N человек, пронумерованных от 1 до N. При ведении счета по кругу вычеркивается каждый второй человек, пока не останется один. Составить две программы, моделирующие процесс. Одна из программ должна использовать класс **ArrayList**, а вторая — **LinkedList**. Какая из двух программ работает быстрее? Почему?

import java.util.\*;

public class nomer1 {

public static void main(String[] args) {

// Создаем список из N элементов

Scanner id = new Scanner(System.in);

int N = id.nextInt();

ArrayList<Integer> arrayList = new ArrayList<>();

LinkedList<Integer> linkedList = new LinkedList<>();

for (int i = 1; i <= N; i++) {

arrayList.add(i);

linkedList.add(i);

}

for (int i = 0; i < arrayList.size(); i += 2) {

arrayList.remove(i);

}

for (int i = 0; i < linkedList.size(); i += 2) {

linkedList.remove(i);

}

System.out.println("Оставшийся человек в ArrayList: " + arrayList.get(0));

System.out.println("Оставшийся человек в LinkedList: " + linkedList.getFirst());

}

}

LinkedList будет работать быстрее, так как он не требует перемещения элементов в памяти. ArrayList будет перемещать элементы в памяти, что может привести к снижению производительности.

8. Разработать проект, в котором для ввода, вывода и изменения односвязного линейного списка создать следующие методы:

а) с использованием цикла:

* ввод с головы createHead();
* ввод с хвоста createTail();
* вывод (возвращается строка, сформированная из элементов списка) toString();
* добавление элемента в начало списка AddFirst();
* добавление элемента в конец списка AddLast();
* вставка элемента в список с указанным номером Insert();
* удаление элемента с головы списка RemoveFirst();
* удаление последнего элемента списка RemoveLast();
* удаление из списка элемента с указанным номером
  + - Remove();

б) с использованием рекурсии:

* ввод с головы createHeadRec();
* ввод с хвоста createTailRec();
* вывод (возвращается строка, сформированная из элементов списка) toStringRec().

А)

public class nomer1 {

public static void main(String[] args){

}

private Node head;

private Node tail;

public void LinkedList() {

head = null;

tail = null;

}

public void createHead() {

Node newNode = new Node(0);

newNode.next = head;

head = newNode;

}

public void createTail() {

Node newNode = new Node(0);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

public String toString() {

StringBuilder sb = new StringBuilder();

Node currentNode = head;

while (currentNode != null) {

sb.append(currentNode.data + " ");

currentNode = currentNode.next;

}

return sb.toString();

}

public void AddFirst(int data) {

Node newNode = new Node(data);

newNode.next = head;

head = newNode;

}

public void AddLast(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

public void Insert(int data, int index) {

if (index == 0) {

AddFirst(data);

} else if (index == size() - 1) {

AddLast(data);

} else {

Node newNode = new Node(data);

Node currentNode = head;

for (int i = 0; i < index - 1; i++) {

currentNode = currentNode.next;

}

newNode.next = currentNode.next;

currentNode.next = newNode;

}

}

public void RemoveFirst() {

if (head == null) {

return;

}

head = head.next;

}

public void RemoveLast() {

if (head == null) {

return;

}

Node currentNode = head;

while (currentNode.next != null && currentNode.next.next != null) {

currentNode = currentNode.next;

}

tail = currentNode;

tail.next = null;

}

public void Remove(int index) {

if (index == 0) {

RemoveFirst();

} else if (index == size() - 1) {

RemoveLast();

} else {

Node currentNode = head;

for (int i = 0; i < index - 1; i++) {

currentNode = currentNode.next;

}

currentNode.next = currentNode.next.next;

}

}

public int size() {

Node currentNode = head;

int size = 0;

while (currentNode != null) {

size++;

currentNode = currentNode.next;

}

return size;

}

private class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

}

Б)

public class nomer1 {

public static void main(String[] args){

}

public void createHeadRec(Node head, Object element) {

if (head == null) {

head = new Node(element);

} else {

head.setNext(new Node(element));

}

}

public void createTailRec(Node head, Object element) {

if (head == null) {

head = new Node(element);

} else {

Node current = head;

while (current.getNext() != null) {

current = current.getNext();

}

current.setNext(new Node(element));

}

}

public String toStringRec(Node head) {

if (head == null) {

return "";

} else {

return head.getElement() + ", " + toStringRec(head.getNext());

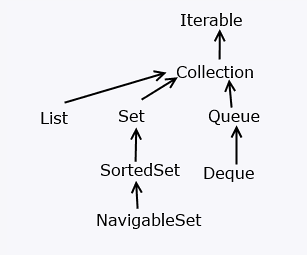
}

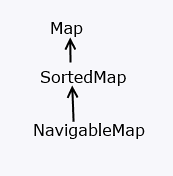
}

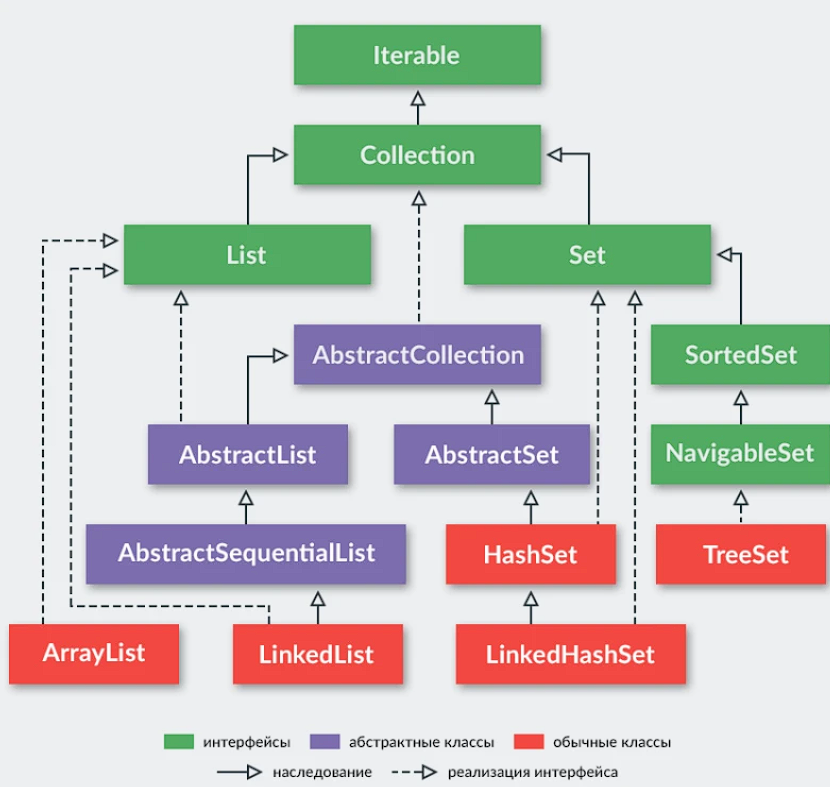
}

# 4 ОБЩЕЕ ОПИСАНИЕ КОЛЛЕКЦИЙ В JAVA

Краткое описание интерфейса Collections и Map, необходимо привести изображения (минимум 3 рисунка), а также две таблицы описывающие основные методы, используемые в интерфейсах Collections и Map.







|  |  |
| --- | --- |
| Методы | Описание |
| addAll(colls, e1, e2, e3, ..) | Добавляет в коллекцию colls элементы e1, e2, e3,... |
| fill(list, obj) | Заменяет в переданном списке все элементы на obj |
| nCopies(n, obj) | Возвращает список, состоящий из n копий объекта obj |
| replaceAll(list, oldVal, newVal) | Заменяет в списке list все значения oldVal на newVal |
| copy(dest, src) | Копирует все элементы из списка src в список dest |
| reverse(list) | Разворачивает список задом наперед |
| sort(list) | Сортирует список в порядке возрастания |
| rotate(list, n) | Циклично сдвигает элементы списка list на n элементов |
| shuffle(list) | Случайно перемешивает элементы списка |
| min(colls) | Находит минимальный элемент коллекции colls |
| max(colls) | Находит максимальный элемент коллекции colls |
| frequency(colls, obj) | Определяет, сколько раз элемент obj встречается в коллекции colls |
| binarySearch(list, key) | Ищет элемент key в отсортированном списке, возвращает индекс. |
| disjoint(colls1, colls2) | Возвращает true, если у коллекций нет общих элементов |

|  |  |
| --- | --- |
| keySet() | возвращает множество(Set) ключей |
| values() | возвращает коллекцию(Collection) значений |
| entrySet() | возвращает множество(Set) наборов “ключ-значение” |
| keySet() | возвращает множество(Set) ключей |

# 5 ОПИСАНИЕ КОЛЛЕКЦИЙ

# (в соответствии с вариантом)

Приведите описание реализации конкретной коллекции. (как работает механизм добавления элементов, как работает механизм удаления элементов, как работает механизм поиска элементов и т.д.)

Напишите конкретную вычислительную сложность добавления, удаления и получения элементов в каждой коллекции в соответствии с вариантом;

Каким образом хранятся элементы в коллекции? (упорядочено или хаотично и т.п.)

ArrayDeque - это реализация коллекции в Java, которая представляет собой двустороннюю очередь. Она поддерживает операции добавления и удаления элементов с обоих концов очереди.

Механизм добавления элементов в ArrayDeque работает следующим образом:

1. Если размер очереди меньше максимального размера, то новый элемент добавляется в конец очереди.

2. Если размер очереди равен максимальному размеру, то новый элемент добавляется в начало очереди, а последний элемент удаляется.

Механизм удаления элементов из ArrayDeque работает следующим образом:

1. Если размер очереди больше нуля, то элемент удаляется из конца очереди.

2. Если размер очереди равен нулю, то операция удаления не может быть выполнена.

Меанизм поиска элементов в ArrayDeque работает следующим образом:

1. Если элемент, который нужно найти, находится в начале очереди, то он возвращается.

2. Если элемент, который нужно найти, находится в конце очереди, то он возвращается.

3. Если элемент, который нужно найти, не находится в начале или в конце очереди, то выполняется поиск элемента в середине очереди.

TreeMap - это реализация коллекции в Java, которая представляет собой ассоциативный массив, где ключи и значения хранятся в отсортированном порядке.

Механизм добавления элементов в TreeMap работает следующим образом:

1. Если ключ уже существует в TreeMap, то новый элемент добавляется с тем же ключом, а старый элемент заменяется новым.

2. Если ключ не существует в TreeMap, то новый элемент добавляется с новым ключом.

Механизм удаления элементов из TreeMap работает следующим образом:

1. Если ключ существует в TreeMap, то элемент удаляется из TreeMap.

2. Если ключ не существует в TreeMap, то операция удаления не может быть выполнена.

Механизм поиска элементов в TreeMap работает следующим образом:

1. Если ключ, который нужно найти, существует в TreeMap, то он возвращается.

2. Если ключ, который нужно найти, не существует в TreeMap, то возвращается null.

ArrayList - это реализация коллекции в Java, которая представляет собой динамический массив.

Механизм добавления элементов в ArrayList работает следующим образом:

1. Если размер ArrayList меньше максимального размера, то новый элемент добавляется в конец ArrayList.

2. Если размер ArrayList равен максимальному размеру, то размер ArrayList увеличивается, и новый элемент добавляется в конец ArrayList.

Механизм удаления элементов из ArrayList работает следующим образом:

1. Если индекс элемента, который нужно удалить, больше нуля и меньше размера ArrayList, то элемент удаляется из ArrayList.

2. Если индекс элемента, который нужно удалить, равен нулю или больше размера ArrayList, то операция удаления не может быть выполнена.

Механизм поиска элементов в ArrayList работает следующим образом:

1. Если индекс элемента, который нужно найти, больше нуля и меньше размера ArrayList, то элемент возвращается.

2. Если индекс элемента, который нужно найти, равен нулю или больше размера ArrayList, то возвращается null.

Таким образом, ArrayList является эффективной реализацией коллекции для операций добавления, удаления и поиска элементов.

# 

# 6 СРАВНЕНИЕ КОЛЛЕКЦИЙ

# (в соответствии с вариантом)

Необходимо произвести вычисление времени работы нижеперечисленных операций, количество элементов коллекциях и сами коллекции указаны в индивидуальном варианте. После выполнения вычислений необходимо заполнить таблицы и привести скриншоты кода программы и данных вывода в консоль.

1. Сравнение времени выполнения операции добавление элемента в начало коллекции;
2. Сравнение времени выполнения операции добавление элемента в конец коллекции;
3. Сравнение времени выполнения операции добавление элемента в середину коллекции;
4. Сравнение времени выполнения операции удаления элемента в начале коллекции;
5. Сравнение времени выполнения операции удаления элемента в конце коллекции;
6. Сравнение времени выполнения операции удаления элемента в середине коллекции;
7. Сравнение времени выполнения операции получения элемента по индексу;

Листинг 1. Пример кода для вычисления времени добавления n количества элементов в коллекции

package lr10.example8;  
  
import java.util.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 List<Integer> arrayList = new ArrayList<>();  
 List<Integer> linkedList = new LinkedList<>();  
  
 System.*out*.println("Время выполнения операции добавления arrayList = " + *getRunningTime*(arrayList));  
 System.*out*.println("Время выполнения операции добавления linkedList = " + *getRunningTime*(linkedList));  
  
 }  
  
 private static long getRunningTime(List<Integer> list){  
  
 *// точка начала отсчета времени выполнения программы* long start = System.*currentTimeMillis*();  
  
 *// блок кода в котором выполняется операция добавления* for (int i = 0; i < 1000000; i++) {  
 list.add(i);  
 }  
  
 *// точка окончания отсчета времени выполнения программы* long end = System.*currentTimeMillis*();  
  
 *// вывод в консоль времени выполнения блока кода содержащего операцию* return end - start;  
  
 }  
}

Пример таблицы 1.

Таблица 1. Сравнение времени выполнения операции добавления

|  |  |  |  |
| --- | --- | --- | --- |
| Коллекция | в начало коллекции | в середину коллекции | в конец коллекции |
| ArrayList | 363 | 407 | 370 |
| ArrayDeque | 336 | 364 | 319 |
| TreeMap | 2114 | 2329 | 2039 |

Таблица 2. Сравнение времени выполнения операции удаления

|  |  |  |  |
| --- | --- | --- | --- |
| Коллекция | в начале коллекции | в середине коллекции | в конце коллекции |
| ArrayList | 147 | 161 | 133 |
| ArrayDeque | 124 | 149 | 128 |
| TreeMap | 1445 | 1567 | 1497 |

Таблица 3. Сравнение времени выполнения операции получения элемента по индексу

|  |  |
| --- | --- |
| Коллекция | По индексу для <n> элементов |
| ArrayList | 97 |
| ArrayDeque | 86 |
| TreeMap | 1027 |

# 7 ОТВЕТЫ НА ВОПРОСЫ

# (в соответствии с вариантом)

Вопросы приведены в описании вариантов

**Варианты**

Количество элементов в коллекции:

<номер варианта \* 1 000 000>

Количество элементов для вычисления скорости получения по индексу: <номер варианта \* 1 000 000 000>

|  |  |
| --- | --- |
| Номер варианта | Коллекции |
| 1 | ArrayDeque, LinkedList, ArrayList |
| 2 | ArrayList, SortedSet, LinkedList |
| 3 | ArrayDeque, LinkedHashSet, ArrayList |
| 4 | ArrayList, TreeSet, LinkedHashMap |
| 5 | ArrayDeque, ArrayList, SortedSet |
| 6 | ArrayList, LinkedHashMap, LinkedList |
| 7 | ArrayDeque, ArrayList, LinkedList |
| 8 | HashSet, SortedSet, ArrayList |
| 9 | ArrayDeque, ArrayList, TreeSet |
| 10 | HashSet, LinkedHashMap, ArrayList |
| 11 | ArrayDeque, SortedMap, ArrayList |
| 12 | ArrayList, TreeMap, ArrayDeque |
| 13 | ArrayDeque, LinkedList, TreeSet |
| 14 | ArrayList, LinkedHashSet, LinkedHashMap |
| 15 | ArrayDeque, TreeSet, ArrayList |
| 16 | ArrayList, LinkedHashMap, SortedSet |
| 17 | ArrayDeque, TreeMap, ArrayList |
| 18 | ArrayList, LinkedList, TreeSet |
| 19 | ArrayDeque, SortedSet, SortedMap |
| 20 | ArrayList, SortedSet, TreeSet |

Вопросы:

1. В чем разница между fail-safe и fail-fast свойствами?
2. В чем разница между Iterator и Enumeration?
3. В чем разница между Iterator и ListIterator?
4. Дайте определение понятию “итератор”.
5. Как задается порядок следования объектов в коллекции, как отсортировать коллекцию?
6. Как получить коллекцию только для чтения?
7. Как получить не модифицируемую коллекцию?
8. Как получить синхронизированную коллекцию из не синхронизированной?
9. Как реализован цикл foreach?
10. Какие есть способы перебора всех элементов List?
11. Какие коллекции синхронизированы?
12. Какую функциональность представляет класс Collections?
13. Почему Map не наследуется от Collection?
14. Почему в классе iterator нет метода для получения следующего элемента без передвижения курсора?
15. Почему нет метода iterator.add() чтобы добавить элементы в коллекцию?
16. Расскажите о реализации классов HashSet и TreeSet?
17. Чем отличаются ArrayDequeи TreeMap? Как они устроены и работают? Что со временем доступа к объектам, какие зависимости?
18. Что будет, если в Map положить два значения с одинаковым ключом?
19. Что делать, чтобы не возникло исключение ConcurrentModificationException?
20. Что такое Hashtable, чем она отличается от HashMap? На сегодняшний день она deprecated, как все-таки использовать нужную функциональность?

1. Fail-safe и fail-fast свойства определяют поведение итератора при изменении коллекции во время итерации. Fail-safe итераторы продолжают работать корректно, даже если коллекция изменяется во время итерации. Fail-fast итераторы останавливают итерацию и выбрасывают исключение ConcurrentModificationException, если коллекция изменяется во время итерации.

2. Iterator и Enumeration - это интерфейсы, которые позволяют перебирать элементы коллекции. Основное отличие между ними заключается в том, что Iterator предоставляет больше методов и более гибкий в использовании. Enumeration, с другой стороны, имеет меньше методов и не позволяет изменять коллекцию во время итерации.

3. Iterator и ListIterator - это интерфейсы, которые позволяют перебирать элементы коллекции. Iterator используется для перебора элементов в коллекции в прямом порядке, в то время как ListIterator позволяет перебирать элементы в прямом и обратном порядке, а также изменять элементы.

4. Итератор - это объект, который позволяет перебирать элементы коллекции. Он предоставляет методы для получения следующего элемента, проверки, есть ли еще элементы, и изменения состояния итератора.

5. Порядок следования объектов в коллекции определяется методом, который используется для добавления элементов в коллекцию. Коллекции, такие как ArrayList, поддерживают порядок следования элементов, который можно изменить с помощью метода sort().

6. Чтобы получить коллекцию только для чтения, можно использовать метод unmodifiableCollection() класса Collections. Этот метод возвращает немодифицируемую коллекцию, которая не позволяет изменять элементы или размер коллекции.

7. Чтобы получить немодифицируемую коллекцию, можно использовать метод unmodifiableCollection() класса Collections. Этот метод возвращает коллекцию, которая не позволяет изменять элементы или размер коллекции.

8. Чтобы получить синхронизированную коллекцию из не синхронизированной, можно использовать метод synchronizedCollection() класса Collections. Этот метод возвращает синхронизированную коллекцию, которая предотвращает одновременное изменение коллекции несколькими потоками.

9. Цикл foreach реализован с помощью итератора. Он использует итератор для перебора элементов коллекции и выполнения действий с каждым элементом.

10. Есть несколько способов перебора всех элементов List. Один из способов - использовать цикл for-each, который автоматически создает итератор и перебирает все элементы. Другой способ - использовать итератор, созданный с помощью метода listIterator() класса List.

11. Коллекции, такие как Vector и Hashtable, являются синхронизированными. Это означает, что они обеспечивают безопасное использование несколькими потоками.

12. Класс Collections предоставляет множество полезных статических методов для работы с коллекциями, таких как сортировка, поиск, копирование и преобразование коллекций.

13. Map не наследуется от Collection, потому что Map и Collection имеют разные функциональные возможности и требования. Map представляет собой ассоциативную коллекцию, которая позволяет хранить пары ключ-значение, в то время как Collection представляет собой неупорядоченную коллекцию объектов.

14. В классе Iterator нет метода для получения следующего элемента без передвижения курсора, потому что это противоречит концепции итератора. Итератор предназначен для перебора элементов в определенном порядке, и каждый следующий элемент доступен только после получения предыдущего.

15. В классе Iterator нет метода add() для добавления элементов в коллекцию, потому что это противоречит концепции итератора. Итератор предназначен для перебора элементов в определенном порядке, и добавление элементов во время итерации может привести к непредсказуемым результатам.

16. HashSet и TreeSet - это реализации интерфейса Set в Java. HashSet использует хеш-таблицу для хранения элементов, что обеспечивает быстрый доступ к элементам. TreeSet использует дерево для хранения элементов, что обеспечивает упорядоченность элементов.

17. ArrayDeque и TreeMap - это реализации интерфейсов Deque и Map в Java. ArrayDeque использует массив для хранения элементов, что обеспечивает быстрый доступ к элементам. TreeMap использует дерево для хранения элементов, что обеспечивает упорядоченность элементов. Время доступа к объектам в ArrayDeque и TreeMap зависит от размера коллекции и структуры данных.

18. Если в Map положить два значения с одинаковым ключом, то второе значение заменит первое.

19. Чтобы избежать исключения ConcurrentModificationException, нужно использовать итератор для изменения коллекции.

20. Hashtable - это устаревшая реализация HashMap в Java. Она отличается от HashMap тем, что Hashtable не является потокобезопасной и не поддерживает null ключи. Для использования нужной функциональности Hashtable можно использовать HashMap.