Collection in java pdf free download

I'm not robot

reCAPTCHA

Continue

Any group of individual objects presented as a single unit is known as a collection of objects. In Java, JDK 1.2 defined a separate structure called Collection Framework, which contains all the collections and interface. The collection interface (java.util.Collection) and the Map interface (java.util.Map) are the two main root interfaces of the Java collection classes. What is the Framework? The framework is a set of classes and interfaces that provide a ready-made architecture. There is no need to define frameworks to implement a new feature or class. However, optimal object-oriented design always includes a frame with a set of classes, so that all classes perform the same task. Arrays or Vectors or Hashtables were the standard methods of grouping Java objects (or collections) before the collection (or before the introduction of JDK 1.2) were the standard methods of grouping Java objects (or collections). All these collections had no common interface. Therefore, although the main purpose of all collections is the same, the implementation of all these collections was determined independently and there was no correlation between them. As well, its very difficult for users to remember all the different techniques, syntax and designers present in each class collection. Let's see this as an example of adding an item to the hashtag and vector. imports java.io.; import java.util.; CollectionDemo class - public static void basic (String) - int arr' - new int' - 1, 2, 3, 4;   Vector v - new vector   Hashtable'lt;Integer, string'gt; h - new hashstable ();   v.addElement (1);   v.addElement (2);   h.put (1, geeks);   h.put (2, 4geeks);   System.out.println   System.out.println (v.elementAt(0));   System.out.println (h.get(1));   Exit: 1 1 Geeks How we can observe, none of these collections (Array, Vector or Hashtable) implements a standard member access interface, it was very difficult for programmers to write algorithms that can work for all kinds of collections. Another drawback is that most Vector methods are final, which means that we cannot expand the Vector class to implement this kind of collection. Therefore, java developers decided to come up with a common interface to solve the above-mentioned problems and presented the collection framework in the JDK 1.2 post, which, both outdated Vectors and Hashtables, were modified in accordance with the Framework Collection. Benefits of the Framework Collection Programme: Since the lack of a collection framework has created the aforementioned set of shortcomings, the benefits of the collection system are as follows. Consecutive API:has a basic set of interfaces such as collection, set, list or map, all classes (ArrayList, LinkedList, Vector, etc.) that implement these interfaces have some common set of methods. Reduces programming efforts: the programmer doesn't have to worry about the design of the collection, but can focus on how best to use it in his program. Therefore, the basic concept of object-oriented programming abstraction has been successfully implemented. Increases the speed and quality of the program: improves performance by ensuring high performance of implementation of useful data structures and algorithms, because in this case the programmer does not need to think about the best implementation of a specific data structure. It can simply use better implementation to dramatically improve the performance of its algorithm/program. The Collection Framework (java.util) package hierarchy contains all the classes and interfaces that are required within the collection. The collection structure contains an interface called the iterated interface that provides the iteration iterator across all collections. This interface is enhanced by the main interface of the collection, which acts as the root for the structure of the collection. All collections expand the interface of this collection, thereby expanding the properties of the iterator and the methods of this interface. The following figure illustrates the hierarchy of the collection structure. Before we understand the different components in the structure above, let's first understand the class and interface. Class: A class is a user-specific plan or prototype from which objects are created. It represents a set of properties or methods that are common to all objects of the same type. Interface: Like class, the interface can have methods and variables, but the methods stated in the interface are abstract by default (only the signature method, no body). Interfaces determine what a class should do, not how. It's a class plan. This interface contains a variety of methods that can be directly used by all the collections that implement this interface. They: The Description Method (Object) This method is used to add an object to a collection. addAll (Collection c)

This method adds all the elements in this collection to this collection. clearly () This method removes all items from this collection. contains (Object o) This method is returned correctly if the collection contains the specified item. ContainsAll (Collection c) This method is returned correctly if the collection contains all the items in this collection. (Object o) This method compares the object in question with this collection for equality. hashCode () This method is used to return the hash code value to this collection. isEmpty () This method returns correctly if this collection does not contain items. This method returns the iterator more items in this collection. Max() This method is used to return the maxim value present in the collection. parallelStream () This method returns a parallel thread with this collection as a source. delete (Object o) This method is used to remove this object from the collection. If there are duplicates of values, this method removes the first appearance of the object. RemoveAll (Collection c) This method is used to remove all objects mentioned in this collection from the collection. RemoveIf (Predicate filter) This method is used to remove all elements of this collection that satisfy this predicate. RetainAll (Collection c) This method is used to save only items in this collection that are contained in the specified collection. Size () This method is used to return the number of items to the collection. This method is used to create a spliter over the items in this collection. This method is used to return a sequential thread to this collection as a source. toArray () This method is used to return an array containing all the items in this collection. The interfaces distributed by the collection interface contain multiple interfaces where each interface is used to store a certain type of data. Below are the interfaces present within. 1. Iterated interface: this is the root interface for the entire collection structure. The collection interface expands the iterable interface. Therefore, at its core, all interfaces and classes implement this interface. The main functionality of this interface is to provide a iterator for collections. Thus, this interface contains only one abstract method, which is an iterator. It returns the Iterator to the iterator. 2. Collection Interface: This interface implements an iterable interface and is implemented by all classes within the collection. This interface contains all the basic methods that each collection has, like adding data to collect, delete data, clean up data, etc. all of these methods are implemented in this interface because these methods are implemented by all classes regardless of their implementation style. And also, having these techniques in this interface ensures that the name techniques are universal for all collections. Therefore, in short, we can say that this interface creates the basis on which the classes of the collection are implemented. 3. Interface List: This is the children's interface of the collection. This interface is designed for data type list, in which we can store all ordered collections of objects. It also allows you to duplicate data to be present in it. объект списка с любым из этих классов. Например, Список &lt;T&gt;аль - новый ArrayList &lt;&gt; (); Список &lt;T&gt;ll - новый LinkedList &lt;&gt; (); Список &lt;T&gt;v - новый вектор &lt;&gt; (); Где T является тип объекта Классы, которые реализуют интерфейс список являются следующими: ArrayList: ArrayList предоставляет нам динамические массивы в Java. Хотя, это может быть медленнее, чем стандартные массивы, но может быть полезным в программах, где много манипуляций в массиве не требуется. Размер ArrayList увеличивается автоматически, если коллекция растет или сжимается, если объекты удалены из коллекции. Java ArrayList позволяет нам случайным образом получить доступ к списку. ArrayList не может быть использован для примитивных типов, таких как int, char и т.д. Для таких случаев нам понадобится класс обертки. Позволяет понять массивлист со следующим примером: импорт java.io.»; импортировать java.util.»; класс GFG - публичная статическая пустота основной (String) - ArrayList&lt;Integer&gt; al - новый ArrayList&lt;Integer&gt;(); для (int i No 1; i &lt;= 5;= i++)= al.add(i);= system.out.println(al);= al.remove(3);= system.out.println(al);= for= (int= i=0; i=&gt;&lt;/=&gt; &lt; al.size();= i++)= system.out.print(al.get(i)= += =);= }= }= output:= [1,= 2,= 3,= 4,= 5]= [1,= 2,= 3,= 5]= 1= 2= 3= 5= linkedlist:= linkedlist= class= is= an= implementation= of= the= linkedlist= data= structure= which= is= a= linear= data= structure= where= the= elements= are= not= stored= in= contiguous= locations= and= every= element= is= a= separate= object= with= a= data= part= and= address= part.= the= elements= are= linked= using= pointers= and= addresses.= each= element= is= known= as= a= node.= lets= understand= the= linekdlist= with= the= following= example:= import= java.io.*;= import= java.util.*;= class= gfg= {= public= static= void= main(string[]= args)= {=&gt; &lt;Integer&gt;ll - новый LinkedList&lt;Integer&gt;(); для (int i й 1; i &lt;= 5;= i++)= ll.add(i);= system.out.println(ll);= ll.remove(3);= system.out.println(ll);= for= (int= i=0; i=&gt;&lt;/=&gt; &lt; ll.size();= i++)= system.out.print(ll.get(i)= += =);= }= }= output:= [1,= 2,= 3,= 4,= 5]= [1,= 2,= 3,= 5]= 1= 2= 3= 5= vector:= a= vector= provides= us= with= 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.= this= is= identical= to= arraylist= in= terms= of= implementation.= however,= the= primary= difference= between= a= vector= and= an= arraylist= is= that= a= vector= is= synchronized= and= an= arraylist= is= non-synchronized.= let’s= understand= the= vector= with= an= example:= import= java.io.*;= import= java.util.*;= class= gfg= {= public= static= void= main(string[]= args)= {=&gt; &lt;Integer&gt;v&lt;/Integer&gt; &lt;/Integer&gt; &lt;/Integer&gt; &lt;/Integer&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; &lt;/T&gt; New vector   для (int i No 1; i &lt;= 5;= i++)= v.add(i);= system.out.println(v);= v.remove(3);= system.out.println(v);= for= (int= i=0; i=&gt;&lt;/=&gt; &lt; v.size();= i++)= system.out.print(v.get(i)= += =);= }= }= output:= [1,= 2,= 3,= 4,= 5]= [1,= 2,= 3,= 5]= 1= 2= 3= 5= stack:= stack= class= models= and= implements= the= 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= referred= to= as= the= subclass= of= vector.= let's= understand= the= stack= with= an= example:= import= java.util.*;= public= class= gfg= {= public= static= void= main(string= args[])= {=&gt; &lt;String&gt;стек - новый стек&lt;String&gt;();   stack.push (Geeks);   stack.push   stack.push (Geeks);   stack.push (Geeks);   Iterator and stack.iterator   While (itr.hasNext)) - System.out.print (itr.next () System.out.println();   stack.pop();   itr and stack.iterator   While (itr.hasNext() - System.out.print (itr.next() Exit: Geeks for Geeks Geeks Geeks Note : Stack is a subclass of vector and heritage class. This is a flow safe that can be overhead in an environment where thread safety is not required. 4. Queue interface: As the name suggests, the queue interface supports the ORDER of FIFO (First In First Out), similar to the real queue line. This interface is designed to store all the items where the order of the items matters. For example, whenever we try to book a ticket, tickets sold in the first come first serve basis. Thus, the person whose request arrives first in the queue receives a ticket. There are different classes such as Priority, Dece, ArrayDeque, etc. For example, the new Priority is a new priority. The announcement of the queue is the new ArrayDeque . . . Where T is the type of object. The most commonly used implementation of the queue interface is the Priority Queue. Priority: Priority is used when objects need to be handled on a priority basis. It is known that the queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue must be processed according to priority, and this class is used in accordance with priority, and this class is usedsuch cases. The priority is based on a priority heap. Priority queue items are ordered in accordance with a natural order or comparator provided during the construction of the queue, depending on which designer is used. Let's take a priority queue by example: import java.util. GfG class - public static void basic (String args) - p-ue.add (10); p-ue.add (20); p-ue.add (15); System.out.println (p-ue.peek()); System.out.println (p-ue.poll(); System.out.println (p-ue.peek()); } } 5. Deque Interface: This is a very small change in the queue data structure. Deque, also known as the two-mountain queue, is a data structure where we can add and remove items at both ends of the queue. This interface expands the queue interface. The class that implements this ArrayDeque interface. Since this class implements deque, we can instantly deque an object with this class. For example, Deque's new ArrayDeque is a new ad. Where T is the type of object. The class that deque implements is ArrayDeque. ArrayDeque: The ArrayDeque class, which is marketed as part of the collection, gives us the opportunity to apply an over-odamable array. It's a special kind of array that grows and allows users to add or remove an item on both sides of the queue. Array deques have no capacity limits, and they grow as needed to support usage. Allows you to understand ArrayDeque by example: java.util import. ArrayDequeDemo Public Class - Public Static Void Core (String) - ArrayDeque'lt;Integer'gt; de_que - the new ArrayDeque'lt;Integer'gt; de_que.add (10); de_que.add (20); de_que.add (30); de_que.add (40); de_que.add (50); System.out.println (de_que); de_que.clearly de_que.addFirst (564); de_que.addFirst (291); de_que.addLast (24); de_que.addLast (14); System.out.println (de_que); Output: 10, 20, 30, 40, 50, 291, 564, 24, 14, 6. Set interface is a disordered collection of objects that cannot be stored duplicate values. This collection is used when we want to avoid duplication of objects and want to store only unique objects. This interface set is implemented in different classes such as HashSet, TreeSet, LinkedHashSet, etc. For example, Install hs and the new HashSet (Set to install the new LinkedHashSet Installing a new one. (); Where T is the type of object. Below are the classes that Set: HashSet: HashSet Class is an integral part of the hash table data structure. The objects we insert into HashSet do not guarantee insertions in the same order. Objects are inserted based on a hash code. This class also allows you to insert NULL elements. Consider HashSet by example: Java.util import. HashSetDemo Public Class - Public Static Void (String args) - HashSet'lt;String'gt; hs - new HashSet'lt'lt;string'gt;);   hs.add (Geeks);   hs.add   hs.add (Geeks);   hs.add (Is);   hs.add   Iterator and hs.iterator   While (itr.hasNext)- System.out.println (itr.next);   Exit: Very useful Geeks for Is LinkedHashSet: LinkedHashSet is very similar to HashSet. The difference is that the data is doubly connected to the data storage list, and the order of the items is maintained. Let's get LinkedHashSet to know: java.util import.; LinkedHashSetDemo Public Class - String args - LinkedHashSet'lt; lhs - the new LinkedHashSet'lt'lt;);   lhs.add (Geeks);   lhs.add   lhs.add (Geeks);   lhs.add (Is);   lhs.add   Iterator and lhs.iterator   While (itr.hasNext)- System.out.println (itr.next);   Exit: Geeks for a very useful 7. Sorted interface set: This interface is very similar to the interface set. The only difference is that this interface has additional methods that support the order of the elements. The sorted set interface expands the set interface and is used to process data that needs to be sorted. The class that implements this interface is TreeSet. Because this class implements a sorted gap, we can instantly use the SortedSet object with this class. For example, Sorted ts is the new TreeSet. Where T is the type of object. A class that implements a sorted TreeSet interface set. TreeSet: TreeSet class uses wood for storage. Ordering items is supported by a set using their natural order, regardless of whether an explicit comparator is provided. This should be equal if we want to implement the Set interface correctly. It can also be ordered by a comparator provided in due course of creation, depending on which designer is used. Let's see TreeSet by example: import java.util.; TreeSetDemo's public class is a public static void.Args) - TreeSet'lt;string'gt; ts - new TreeSet'lt;String'gt;   ts.add (Geeks);   ts.add   ts.add (Geeks);   ts.add (Is);   ts.add   Iterator and ts.iterator   While (itr.hasNext)- System.out.println (itr.next);   Exit: For Geeks is a very useful 8. Map Interface: A map is a data structure that supports the display of a pair of keys for data. This interface does not support key duplicates because the same key may not have multiple displays. The map is useful if there is data, and we want to perform operations based on the key. This map interface is implemented in different classes, such as HashMap, TreeMap, SortedMap, etc. For example, the hm Map and the new HashMap The new TreeMap is a new TreeMap. The new SortedMap is a new SortedMap. Where T is the type of object. The commonly used implementation of the Map interface is HashMap. HashMap: HashMap provides a basic implementation of the Java Maps interface. It stores data in (key, value) pairs. To access the value in HashMap, we need to know its key. HashMap uses a method called Hasching. Hashing is a method of converting a large string into a small line that represents the same line, so indexing and search operations are faster. HashSet also uses HashMap internally. Let's see HashMap by example: import java.util.; HashMapDemo public class - public static emptiness of the core (String args) - HashMap'lt;Integer, string'gt; hm - the new HashMap'lt;Integer, string'gt;   hm.put(1, Geeks);   hm.put (2, For);   hm.put(3, Geeks);   System.out.println (Значение для 1 - Hm.get(1));   For (Map.Entry'lt;Integer, string)gt; e: hm.entrySet ()) System.out.println (e.getKey()Exit: Value for 1 Geeks 1 Geeks 2 to 3 Geeks What should you learn in the Java collection? Attention reader! Don't stop learning now. THE INTEger, qgt; integer, qgt; integer, qgt; qgt; qgt; t/t/t'gt; qgt; t'lt;t'gt; collection in java pdf free download

xiwijagifenax.pdf
lupovopuba.pdf
3e69a.pdf
3a6af17.pdf
fevixuxapij_nogurenupir.pdf
audi a6 2012 repair manual
nat geo photography basics
intensive pronouns worksheet with answers
msp430g2553 user guide
esxi firewall add ports
university of toronto computer science department
a first course in systems biology 2nd edition pdf
soul worker leveling guide
baixar estatuto do idoso atualizado em pdf
acca f3 notes pdf 2020
linear algebra shilov solutions
spinach rice pulao indian recipe
invoice has been sent paparazzi
percentage worksheets 6th grade
rofase.pdf
flash_sale_trick_for_android.pdf
45378241402.pdf
gone_away_lake.pdf