Module 1-The collections and Framework

1. **Explainbriefaboutcollectionframework.**
   - TheJavaCollectionsFrameworkstandardizesthewayinwhichgroupsof objects are handled by your programs.
   - Theframeworkhadtobe high-performance.
   - Theimplementationsforthefundamentalcollections(dynamicarrays,linked lists, trees, and hash tables) are highly efficient.
   - Theframeworkhadtoallowdifferenttypesofcollectionstoworkinasimilar manner and with a high degree of interoperability.
   - Extendingand/oradaptingacollection had tobeeasy.
   - Mechanismswereaddedthatallowtheintegrationofstandardarraysintothe Collections Framework.
   - Algorithmsareanotherimportantpartofthecollection mechanism.
   - Algorithmsoperateoncollectionsandaredefinedasstaticmethodswithinthe Collections class.
   - An iterator offers a general-purpose, standardized way of accessing the elementswithinacollection,oneatatime.Thus,aniteratorprovidesameans of enumerating the contents of a collection.
   - BecauseeachcollectionimplementsIterator,theelementsofanycollection class can be accessed through the methods defined by Iterator.

2. **Whataretherecentchangestocollectionframework?**

Recently,theCollectionsFrameworkunderwentafundamental changethat significantlyincreased its power and streamlined its use. The changes were the addition of generics, autoboxing/unboxing, and the for-each style for loop.

Generics

TheadditionofgenericscausedasignificantchangetotheCollectionsFramework because the entire Collections Framework has been reengineered for it. All collections are now generic, and many of the methods that operate on collections take generic type parameters Generics add the one feature that collections had been missing: type safety.

Prior to generics, all collections stored Object references, which meant that any collection couldstoreanytypeofobject.Thus,itwaspossibletoaccidentallystorein compatibletypes in a collection.

Doingsocouldresultinrun-timetypemismatcherrors.Withgenerics,itispossibleto explicitlystatethetypeofdatabeingstored, andrun-timetypemismatcherrorscanbe avoided.

Autoboxing/unboxing

Autoboxingfacilitates theUseof PrimitiveTypes.

Autoboxing/unboxingfacilitates thestoringofprimitive typesin collections.

Asyouwillsee,acollectioncanstoreonlyreferences,notprimitivevalues. Inthepast,if you wanted to store a primitive value, such as an int, in a collection, you had to manually box it into its typewrapper.

Whenthevaluewasretrieved,itneededtobemanuallyunboxed(byusinganexplicitcast) into its proper primitive type.

Because of autoboxing/unboxing, Java can automatically perform the proper boxing and unboxingneededwhenstoringorretrievingprimitivetypes.Thereisnoneedtomanually perform these operations.

TheFor-Each StyleforLoop

collectioncan be cycled through byuseofthefor-eachstyleforloop.

EarlieritwasdonewithIteratableinterface.For eachloopiseasierthanthe earlier iterator.

3. **ListtheCollection Interfaces?**
   - The Collections Framework defines several interfaces. This section provides anoverviewofeachinterface.Collectionenablesyoutoworkwithgroupsof objects; it is at the top of the collections hierarchy.
   - Dequeextends Queueto handleadouble-ended queue.
   - ListextendsCollectiontohandlesequences
   - NavigableSetextendsSortedSettohandleretrievalofelementsbasedon closest-match searches.
   - QueueextendsCollectiontohandlespecialtypes oflistsinwhichelements are removed only from the head.
   - SetextendsCollectiontohandlesets,which mustcontainuniqueelements.
   - SortedSetextendsSettohandlesorted sets.
4. **Givethesyntaxofcollectioninterface.Explainthemethodspresentincollection interface.**

interfaceCollection<E>

Especifiesthetypeofobjectsthatthecollection

Collection extends the Iterable interface.

Iteratingthroughthelistcanebedonethroughthe iteratable interface.

Methodsincollectioninterface

add

booleanadd(Eobj)

addsobj to theinvokingcollection.

Returnstrueifobj was added to thecollection.

Returnsfalseifobjisalreadyamemberofthecollectionand the collection does not allow duplicates.

addAll

booleanaddAll(Collection<?extendsE>c)

Addsallthe elementsofcto theinvokingcollection.

Returns trueif theoperation succeeded

(i.e.,theelementswereadded).Otherwise,returnsfalse.

clear

voidclear()

Removesallelements fromtheinvokingcollection.

contains

booleancontains(Objectobj)

Returnstrueifobjisanelementoftheinvokingcollection.

Otherwise, returns false.

containsAll

booleancontainsAll(Collection<?>c)

Returnstrueiftheinvokingcollectioncontainsallelementsofc. Otherwise, returns false.

equals

booleanequals(Objectobj)

Returnstrueiftheinvokingcollectionandobjareequal. Otherwise,

returns false.

hashCode

inthashCode( )Returnsthe hash code for theinvoking collection.

isEmpty

booleanisEmpty()

Returnstrueiftheinvokingcollectionisempty.

Otherwise, returns false.

iterator

Iterator<E>iterator( ) Returnsaniteratorfor theinvoking collection.

remove

booleanremove(Objectobj)

Removes one instance of obj from the invoking

collection.Returnstrueiftheelementwasremoved.Otherwise,returns

false.

removeAll

booleanremoveAll(Collection<?>c)

Removesallelements ofcfrom theinvokingcollection.

Returnstrueifthecollectionchanged(i.e.,elementswereremoved). Otherwise, returns false.

retainAll

booleanretainAll(Collection<?>c)

Removesallelements fromtheinvokingcollectionexcept thosein c.

Returnstrueifthecollectionchanged(i.e.,elementswereremoved). Otherwise, returns false.

size

intsize()Returnsthenumberofelementsheldintheinvoking collection.

toArray

Object[]toArray()

Returnsanarraythatcontainsalltheelementsstoredinthe invoking collection.

Thearrayelementsarecopiesofthecollectionelements. The

arrayelements are copies of the collection elements.

Ifthesizeofarrayequalsthenumberofelements,theseareturnedin array.

5. ExplainthemethodspresentinListinterface.

Listinterfaceextendscollectioninterface.Itincludesnewmethod.Whichare given below.

voidadd(intindex,Eobj)

Insertsobjinto theinvokinglist attheindexpassedin index.

Anypreexistingelements at or beyond the pointof insertion areshiftedup.

booleanaddAll(intindex,Collection<?extends E>c)

Insertsallelements ofCinto theinvokinglist atthe indexpassed in

index. Anypreexistingelementsatorbeyondthepointofinsertionareshiftedup. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.

Eget(intindex)

Returnstheobjectstoredatthespecifiedindexwithintheinvokingcollection. int

indexOf(Object obj)

Returnstheindexofthefirstinstanceofobjintheinvokinglist.Ifobjisnot an element of the list, –1 is returned.

intlastIndexOf(Objectobj)

Returnstheindexofthelastinstanceofobjintheinvokinglist.Ifobjisnot an element of the list, –1 is returned.

ListIterator<E>listIterator()

Returnsaniteratorto the startof theinvokinglist.

ListIterator<E>listIterator(intindex)

Returnsaniteratortothe invokinglistthatbeginsatthespecifiedindex.

Eremove(intindex)

Removestheelementatpositionindex fromtheinvokinglistandreturnsthedeleted element. The resultinglist is compacted. That is, the indexes of subsequent elements are decremented by one.

Eset(int index, Eobj)

Assigns obj to thelocation specified byindexwithin the invoking list.

List<E>subList(intstart,intend)

Returnsalistthatincludeselementsfromstarttoend –1intheinvokinglist. Elements in the returned list are also referenced by the invoking object.

6. **ExplainSetInterface andset method:**

TheSetinterfacedefines aset.

ItextendsCollectionanddeclaresthebehaviourofacollectionthatdoesnotallow duplicate elements.

Therefore,theadd()methodreturnsfalseifanattempt. Set is

a generic interface that has this declaration:

interfaceSet<E>

Here,Especifiesthetypeofobjectsthatthesetwillhold. The

SortedSet Interface

TheSortedSetinterfaceextendsSetanddeclaresthebehaviorofasetsortedin ascending order.

SortedSetisagenericinterfacethathasthisdeclaration:interfaceSortedSet<E>Here, E specifies the type of objects that the set will hold.

InadditiontothosemethodsdefinedbySet,theSortedSetinterfacedeclaresthe methods.

Comparator<?superE>comparator()

Returnstheinvokingsortedset’s comparator.

Ifthenaturalorderingisusedforthisset,nullisreturned. E

first( )

Returnsthefirst elementintheinvokingsorted set.

SortedSet<E>headSet(Eend)

ReturnsaSortedSetcontainingthoseelementslessthanendthatarecontainedinthe invoking sorted set.

Elementsinthereturnedsortedsetarealsoreferencedbytheinvokingsortedset. E last( )

Returnsthelastelementintheinvokingsortedset.

SortedSet<E> subSet(E start , E end)

ReturnsaSortedSetthatincludesthoseelementsbetweenstartandend–1.Elements in the returned collection are also referenced by the invoking object.

SortedSet<E>tailSet(Estart)

ReturnsaSortedSetthatcontainsthoseelementsgreaterthanorequaltostartthatare contained in the sorted set. Elements in the returned set are also referenced by the invoking object.

SeveralmethodsthrowaNoSuchElementExceptionwhennoitemsarecontainedin the invoking set.

AClassCastExceptionisthrownwhenanobjectisincompatiblewiththeelementsin a set.

ANullPointerExceptionisthrownifanattemptismadetouseanullobjectandnull is not allowed in the set.

AnIllegalArgumentExceptionisthrownifaninvalidargumentisused.

7. **NavigableSetInterfaceandmethod**

The NavigableSet interfaceextends SortedSet and declares the behavior of a collectionthatsupportstheretrievalofelementsbasedontheclosestmatchtoagivenvalue or values.

NavigableSetisagenericinterfacethathasthisdeclaration: interface NavigableSet<E>

Here,Especifiesthetypeofobjectsthatthesetwillhold. NavigableSet

adds the following

Eceiling(E obj)

Searchesthesetforthesmallest element

Ifsuchan elementisfound,itisreturned. Otherwise,nullisreturned.

Iterator<E>descendingIterator()

Returnsaniteratorthatmovesfromthegreatesttoleast.Inotherwords,itreturnsa reverse iterator.

NavigableSet<E>descendingSet()

ReturnsaNavigableSetthatisthereverseoftheinvokingset.Theresultingsetis backed by the invoking set.

Efloor(E obj)

Searchesthesetforthelargestelementesuchthate<=obj.Ifsuchanelementis found, it is returned. Otherwise, null is returned.

NavigableSet<E>headSet(EupperBound,booleanincl)

ReturnsaNavigableSetthatincludesallelementsfromtheinvokingsetthatareless than upperBound . If incl is true, then an element equal to upperBound is included. The resulting set is backed by the invoking set.

Ehigher(Eobj)Searchesthesetforthelargestelementesuchthate>obj.Ifsuchan element is found, it is returned. Otherwise, null is returned.

E lower(E obj)

Searchesthesetforthelargestelementesuchthate<obj.Ifsuchanelementis found, it is returned. Otherwise, null is returned.

EpollFirst()

Returnsthefirstelement,removingtheelementintheprocess.Becausethesetis sorted, this is the element with the least value. null is returned if the set is empty.

E pollLast()

Returns the last element, removing the element in the process. Because the set is sorted,thisistheelementwiththegreatestvalue.nullisreturnedifthesetisempty.

NavigableSet<E>subSet(ElowerBound,booleanlowIncl,EupperBound, boolean highIncl)

ReturnsaNavigableSetthatincludesallelementsfromtheinvokingsetthatare greater than lowerBound and less than upperBound .

If lowIncl is true, then an element equal to lowerBound is included.

IfhighInclistrue,thenanelementequaltoupperBoundisincluded. The

resulting set is backed by the invoking set.

NavigableSet<E>tailSet(ElowerBound,booleanincl)

Returns a NavigableSet that includes all elements from the invoking set that are greaterthanlowerBound.Ifinclistrue,thenanelementequaltolowerBoundis included. The resulting set is backed by the invoking set

8. TheQueueInterfaceand methods

TheQueueinterface extends Collection and declares the behaviour of aqueue, which is oftenafirst-in,first-outlist.However,thereareotypesofqueuesinwhichtheorderingis based upon other criteria. Queue is a generic interface that has this declaration:

interfaceQueue<E>

E element( )

Returnstheelementattheheadofthequeue.Theelementisnotremoved.It throws NoSuchElementException if the queue is empty.

booleanoffer(Eobj)

Attemptstoaddobjtothequeue.Returnstrueifobjwasaddedandfalseotherwise. E peek( )

Returnstheelementattheheadofthequeue. Itreturnsnulliftheequeueisempty.The element is not removed.

E poll( )

Returnstheelementattheheadofthequeue,removingtheelementintheprocess.It returns null if the queue is empty.

Eremove()

Removestheelementattheheadofthequeue,returningtheelementintheprocess.It throws NoSuchElementException if the queue is empty.

9. <u>Dequeinterface</u>

Itextends Queueand declaresthe behaviorofadouble-ended queue.

Double-endedqueuescanfunctionasstandard,first-in,first-outqueuesoras last-in, first-out stacks.

Dequeisagenericinterfacethat hasthis declaration:

interfaceDeque<E>

voidaddFirst(Eobj)

Addsobjtotheheadofthedeque.ThrowsanIllegalStateExceptionifa capacity-restricted deque is out of space.

voidaddLast(Eobj)

Addsobjtothetailofthedeque.ThrowsanIllegalStateExceptionifa capacity-restricted deque is out of space.

Iterator<E>descendingIterator()

Returnsaniteratorthatmovesfromthetailtotheheadofthedeque. Inother words, it returns a reverse iterator.

EgetFirst()

Returnsthefirstelementinthedeque.Theobjectisnotremovedfromthe deque. It throws NoSuchElementException if the deque is empty.

EgetLast()

Returnsthe lastelementin thedeque.

Theobjectisnotremovedfromthedeque.Itthrows NoSuchElementException if the deque is empty.

booleanofferFirst(Eobj)

Attempts to add obj to the head of the deque. Returns true if obj was added andfalseotherwise.Therefore,thismethodreturnsfalsewhenanattemptismadeto add obj to a full, capacity-restricted deque.

booleanofferLast(Eobj)

Attemptstoaddobjtothetailofthedeque.Returnstrueifobjwasaddedand false otherwise.

EpeekFirst()

Returnstheelementattheheadofthedeque. Itreturnsnullifthedequeis empty. The object is not removed.

EpeekLast( )

Returnstheelementatthetailofthedeque. Itreturnsnullifthedequeis empty. The object is not removed.

EpollFirst()

Returnstheelementattheheadofthedeque,removingtheelementintheprocess. It returns null if the deque is empty.

EpollLast()Returnstheelementatthetailofthedeque,removingtheelementinthe process. It returns null if the deque is empty.

Epop()Returnstheelementattheheadofthedeque,removingitintheprocess.It throws NoSuchElementException if the deque is empty.

voidpush(Eobj)Addsobjtotheheadofthedeque.Throwsan IllegalStateException if a capacity-restricted deque is out of space.

EremoveFirst()Returnstheelementattheheadofthedeque,removingtheelement in the process. It throws NoSuchElementException if the deque is empty.

booleanremoveFirstOccurrence(Objectobj)

Removesthefirstoccurrenceofobjfromthedeque.Returnstrueifsuccessfuland false if the deque did not contain obj .

EremoveLast( )

Returnstheelementatthetailofthedeque,removingtheelementintheprocess.It throws NoSuchElementException if the deque is empty.

booleanremoveLastOccurrence(Objectobj)

Removesthelastoccurrenceofobjfromthedeque.Returnstrueifsuccessfuland false if the deque did not contain

10. TheCollectionClasses withexamplecode

AbstractCollection
Implementsmostofthe Collectioninterface.
AbstractList
ExtendsAbstractCollectionandimplementsmostoftheListinterface.
Queue interface.
AbstractSequentialList
ExtendsAbstractListforusebyacollectionthatusessequentialratherthan random access of its elements.
AbstractSet
ExtendsAbstractCollectionandimplements mostoftheSet interface.
EnumSet
ExtendsAbstractSet forusewith enum elements.
HashSet
ExtendsAbstractSetforusewith ahash table.
LinkedHashSet
ExtendsHashSettoallowinsertion-orderiterations.
PriorityQueue
ExtendsAbstractQueueto support apriority-based queue.
TreeSet Implements a set stored in a tree. Extends AbstractSet.
LinkedListImplementsalinkedlistbyextendingAbstractSequentialList.
ArrayList Implements a dynamic array by extending AbstractList.
ArrayDequeImplementsadynamicdouble-endedqueuebyextending AbstractCollection and implementing the Deque interface.
ArrayList

ArrayListclassextends **AbstractList**andimplements the**List** interface

ArrayListisagenericclassthathasthisdeclaration:

class ArrayList<E>

ArrayListhasthe constructorsshownhere:
ArrayList()
constructor builds an empty array list
ArrayList(Collection<?extendsE>*c*)
buildsanarrayliststhat isinitializedwiththeelementsofthecollection*c*.
ArrayList(int*capacity*)

buildsanarrayliststhathasthespecifiedinitial *capacity*.Thecapacityisthe size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an array list.

```
classArrayListDemo{
public static void main(String args[]) {
ArrayList<String>al=newArrayList<String>();
System.out.println("Initial size of al: " +
al.size());
```

```
al.add("C");
al.add("A");
al.add("E");
al.add("B");
al.add("D");
al.add("F");
al.add(1,"A2");
System.out.println("Sizeofalafteradditions:"+ al.size());
System.out.println("Contentsofal:"+al);
al.remove("F");
al.remove(2);
System.out.println("Sizeofalafterdeletions:"+ al.size());
}}
```

ConvertingArrayListtoArray

```
classArrayListToArray{
public static void main(String args[]) {
ArrayList<Integer>al=newArrayList<Integer>();
al.add(1);
al.add(2);
al.add(3);
al.add(4);
System.out.println("Contentsofal:"+al);
Integer ia[] = new Integer[al.size()];
ia=al.toArray(ia);
int sum = 0;
for(int i : ia) sum += i;
System.out.println("Sumis:"+sum);
}
}
```

LinkedList

The**LinkedList** classextends **AbstractSequentialList** andimplementsthe**List**,**Deque**,and **Queue**interfaces.
Itprovidesalinked-listdatastructure.**LinkedList**isagenericclassthat has
this declaration:

```
classLinkedList<E>
```

Here,**E**specifiesthetypeofobjectsthatthelistwillhold. **LinkedList**hasthetwo constructors
LinkedList()
LinkedList(Collection<? extendsE>*c*)
Thefirst constructor builds an emptylinked list.
Thesecondconstructorbuildsalinkedlist thatisinitializedwiththeelementsofthe collection *c*.
Examplecode:

```
import java.util.*;
classLinkedListDemo{
```

```
public static void main(String args[]) {
LinkedList<String>ll=newLinkedList<String>();
ll.add("F");
ll.add("B");
ll.add("D");
ll.add("E");
ll.add("C");
ll.addLast("Z");
ll.addFirst("A");
ll.add(1,"A2");
System.out.println("Originalcontentsofll:"+ll);
ll.remove("F");
ll.remove(2);
System.out.println("Contentsofllafterdeletion:"+ll);
ll.removeFirst();
ll.removeLast();
System.out.println("llafterdeletingfirstandlast:"+ll);
String val = ll.get(2);
ll.set(2, val + " Changed");
System.out.println("llafterchange:"+ll);
}
}
```

HashSet

HashSetextends**AbstractSet**andimplementsthe**Set**interface.Itcreatesacollection that uses a hash table for storage.

JavaHashSetclassis usedtocreate acollectionthatusesahashtable for storage.

ItinheritstheAbstractSetclassandimplementsSet interface.

Theimportantpointsabout JavaHashSetclass are:

- HashSetstorestheelements byusingamechanismcalled**hashing.**
- HashSetcontainsuniqueelementsonly.

HashSetisagenericclassthathasthisdeclaration: class
HashSet<E>
Here,**E**specifies thetypeof objectsthatthesetwill hold.

Constructor

HashSet()
HashSet(Collection<?extendsE>*c*)
HashSet(int *capacity*)
HashSet(int*capacity*,float*fillRatio*)

Example:

```
import java.util.*;
classHashSetDemo{
public static void main(String args[]) {
HashSet<String>hs=newHashSet<String>();
```

```
hs.add("B");
hs.add("A");
hs.add("D");
hs.add("E");
hs.add("C");
hs.add("F");
System.out.println(hs);
}
}
```

output

[D,A,F,C,B, E]

LinkedHashSet

LinkedHashSetclassisaHashtableandLinkedlistimplementationoftheset interface. It inherits HashSet class and implements Set interface.

TheimportantpointsaboutJava LinkedHashSetclassare:

- Contains uniqueelements onlylikeHashSet.
- Providesalloptionalsetoperations,andpermitsnull elements.
- Maintainsinsertionorder.

The**LinkedHashSet**classextends**HashSet**andaddsnomembersofitsown. It is a generic class that has this declaration:

classLinkedHashSet<E>

Here,**E**specifies thetypeof objectsthatthesetwill hold.

LinkedHashSetmaintainsalinkedlistoftheentriesintheset,intheorderinwhich they were inserted.
Thisallowsinsertion-orderiterationovertheset.
Thatis,whencyclingthrougha**LinkedHashSet**usinganiterator,theelementswill be returned in the order in which they were inserted.
Thisis also theorder inwhichtheyarecontainedinthestringreturned by **toString()**whencalledona**LinkedHashSet** object.
Toseetheeffectof**LinkedHashSet**,trysubstituting**LinkedHashSet**for**HashSet**in the preceding program. The output will be

[B,A, D,E, C, F]

whichistheorderin whichthe elementswere inserted.

TreeSet

TreeSetextends**AbstractSet**andimplementsthe**NavigableSet**interface. It creates a collection that uses a tree for storage. Objects are stored in sorted, ascending order. Access and retrieval times are quite fast, which makes **TreeSet** an excellentchoicewhenstoringlargeamountsofsortedinformationthatmustbefound quickly.

TreeSetisagenericclassthathasthisdeclaration: class

TreeSet<E>

Here,**E**specifies thetypeof objectsthatthesetwill hold.

TreeSethasthefollowingconstructors:

TreeSet( )
TreeSet(Collection<? extends E>*c*)
TreeSet(Comparator<?superE>*comp*)
TreeSet(SortedSet<E>*ss*)

Example

```
import java.util.*;
classTreeSetDemo{
public static void main(String args[]) {
TreeSet<String>ts=newTreeSet<String>();
ts.add("C");
ts.add("A");
ts.add("B");
ts.add("E");
ts.add("F");
ts.add("D");
System.out.println(ts);
}
}
```

Theoutput fromthis programis shown here:

[A,B,C, D,E, F]

PriorityQueue

PriorityQueueextends**AbstractQueue**andimplementsthe**Queue**interface. It creates a queue that is prioritized based on the queue's comparator.

PriorityQueueisagenericclassthathasthisdeclaration: class PriorityQueue<E>

Here,**E**specifiesthetypeofobjectsstoredinthequeue.

PriorityQueues are dynamic, growing as necessary.

PriorityQueuedefinesthesixconstructorsshownhere:

PriorityQueue( )
PriorityQueue(int*capacity*)
PriorityQueue(int*capacity*,Comparator<?superE>*comp*)
PriorityQueue(Collection<? extends E>*c*)
PriorityQueue(PriorityQueue<? extends E>*c*)
PriorityQueue(SortedSet<? extends E>*c*)

ArrayDeque

JavaSE6addedthe**ArrayDeque**class,whichextends**AbstractCollection**and implements the **Deque** interface.

Itadds nomethods ofits own.

ArrayDequecreates adynamicarrayand hasno capacityrestrictions.

ArrayDequeisagenericclassthathasthisdeclaration: class ArrayDeque<E>

Here,**E**specifiesthetypeofobjectsstoredinthecollection.

ArrayDeque defines the following constructors:

ArrayDeque( )
ArrayDeque(int *size*)
ArrayDeque(Collection<?extendsE>*c*)

Example:

```
import java.util.*;
classArrayDequeDemo{
public static void main(String args[]) {
ArrayDeque<String>adq=newArrayDeque<String>();
adq.push("A");
adq.push("B");
adq.push("D");
adq.push("E");
adq.push("F");
System.out.print("Poppingthestack:");
while(adq.peek() != null)
System.out.print(adq.pop() + "");
System.out.println();
}
}
```

Theoutput is shown here:

Poppingthe stack: F E DBA

AccessingacollectionViaanIterator:

Beforeyoucanaccessacollectionthroughaniterator,youmustobtainone.Eachof the collection classes provides an **iterator( )** method that returns an iterator to the start of the collection.

Byusingthisiteratorobject, youcanaccesseachelementinthecollection.Element at a time. In general, to use an iterator to cycle through the contents of a collection, follow these steps:

1. Obtainaniteratortothestartofthecollectionbycallingthe collection's**iterator( )** method.
2. Setup aloop thatmakes acall to**hasNext( )**.Havetheloop iterateas longas**hasNext()** returns**true**.
3. Withintheloop, obtaineachelementbycalling**next()**.
4. Forcollectionsthatimplement**List**,youcan alsoobtainaniteratorbycalling **listIterator()**.
5. Asexplained,alistiteratorgives youtheabilitytoaccessthecollectionineitherthe forward or backward direction and lets you modify an element.
6. Otherwise,**ListIterator**is usedjust like**Iterator**.

```
import java.util.*;
classIteratorDemo{
publicstatic void main(Stringargs[]) {

ArrayList<String>al=newArrayList<String>();
al.add("C");
al.add("A");
al.add("E");
```

```
al.add("B");
al.add("D");
al.add("F");

System.out.print("Originalcontentsofal:");

Iterator<String>itr=al.iterator();
while(itr.hasNext()) {
String element = itr.next();
System.out.print(element+"");
}
System.out.println();

ListIterator<String>litr=al.listIterator();
while(litr.hasNext()) {
Stringelement=litr.next();
litr.set(element + "+");
}
System.out.print("Modifiedcontentsofal:"); itr
= al.iterator();
while(itr.hasNext()) {
Stringelement=itr.next();
System.out.print(element +"");
}
System.out.println();
System.out.print("Modifiedlistbackwards:");
while(litr.hasPrevious()) {
Stringelement=litr.previous();
System.out.print(element +"");
}
System.out.println();
}
}
```

Output:
Original contents of al: C A E B D F
Modifiedcontentsofal:C+A+E+B+D+F+
Modifiedlistbackwards:F+D+B+E+A+C+

ForEachloopforiteratingthroughcollection:

```
import
java.util.*;
classForEachDemo{
public static void main(String args[]) {
ArrayList<Integer>vals=newArrayList<Integer>();
vals.add(1);
vals.add(2);
vals.add(3);
vals.add(4);
vals.add(5);
System.out.print("Originalcontentsofvals:"); for(int
v : vals)
```

```
System.out.print(v+"");
System.out.println();
int sum = 0;
for(intv:vals)
sum += v;
System.out.println("Sum ofvalues:"+ sum);
}
}
```

Output:

```
Originalcontentsofvals:12345 Sum of
values: 15
```

StoringUserDefinedClassesin Collections:

collectionsarenot limitedto thestorageof built-in objects.

Thepowerofcollectionsisthattheycanstoreanytypeofobject, including objects of classes that you create.
Userdefinedobjectsstoredin **LinkedList**tostoremailingaddresses:

```
import java.util.*;
class Address {
privateStringname;
privateStringstreet;
private String city;
private String state;
private String code;
Address(Stringn,Strings,Stringc,
String st, String cd) {
name = n;
street = s;
city = c;
state = st;
code=cd;
}
publicStringtoString(){
returnname+"\n"+street+"\n"+ city +
"" + state + "" + code;
}}
classMailList{
public static void main(String args[]) {
LinkedList<Address>ml=newLinkedList<Address>();
ml.add(new Address("J.W. West", "11 Oak Ave",
"Urbana", "IL", "61801"));
ml.add(newAddress("RalphBaker","1142MapleLane",
"Mahomet", "IL", "61853"));
ml.add(newAddress("TomCarlton","867ElmSt",
"Champaign", "IL", "61820"));
for(Address element : ml)
System.out.println(element+"\n");
```

```
System.out.println();
}
}
```

Theoutput fromtheprogramis shown here:

```
J.W. West
11OakAve
Urbana IL61801
Ralph Baker
1142MapleLane
MahometIL61853
Tom Carlton
867 Elm St
ChampaignIL61820
```

RandomAccess Interface:

RandomAccessinterfacecontainsnomembers.

However,byimplementingthisinterface,acollectionsignalsthatitsupportsefficient random access to its elements.

Bycheckingforthe**RandomAccess**interface,clientcodecandetermineatruntimewhether a collection is suitable for certain types of random access operations—especially as they apply to large collections.

RandomAccessis implemented by**ArrayList** andbythe legacy**Vector**class, amongothers.

Working With Maps:

A*map*isan objectthat storesassociations betweenkeysandvalues, or*key/valuepairs.*

Keysandvaluesareobjects.Keysmustbeunique,butthevaluesmaybeduplicated. Some

maps can accept a **null** key and **null** values, others cannot.

There is one key point about maps that is important to mention at the outset: they don't implementthe**Iterable**interface.Thismeansthat you*cannot*cyclethroughamapusinga for-each style **for** loop. Furthermore, you can't obtain an iterator to a map.

However,as youwillsoonsee, youcanobtainacollection-viewofamap,whichdoesallow the use of either the **for** loop or an iterator.

TheMap Interfaces

Becausethemapinterfacesdefinethecharacterandnatureofmaps,thisdiscussionofmaps begins with them.

Thefollowinginterfacessupport maps:

TheMapInterface

The **Map** interface maps unique keys to values. A *key* is an object that you use to retrieve a valueatalaterdate.Givenakeyandavalue, youcanstorethevalueina **Map**object.After the value is stored, you can retrieve it by using its key.

Map is generic:

```
interfaceMap<K,V>
```

Here,**K**specifiesthetypeofkeys,and **V**specifiesthetypeofvalues. The methods declared by **Map.**

Severalmethods throwa**ClassCastException**when anobject is incompatiblewiththeelements inamap.

A**NullPointerException**isthrownifanattemptismadetousea**null**objectand**null**isnot allowed in the map.

An**UnsupportedOperationException**isthrownwhenanattemptis made to change an unmodifiable map.

An**IllegalArgumentException**isthrownifan invalid argument is used.

Mapsrevolvearoundtwobasicoperations: **get()** and**put()**.Toputavalueintoamap, use **put( )**, specifying the key and the value.

Toobtainavalue,call**get()**,passingthekeyas an argument. The value is returned.
maps are not, themselves, collections because they do not implement the **Collection** interface.However, youcanobtainacollection-viewofamap.Todothis,youcanusethe **entrySet( )** method. It returns a **Set** that contains the elements in the map.

Toobtainacollection-viewofthekeys, use **keySet( )**.

Togetacollection-view of thevalues,use**values()**.

Collection-viewsarethemeansbywhichmapsareintegratedintothelargerCollections Framework.

SortedMap
The**SortedMap**interfaceextends**Map**.Itensuresthattheentriesaremaintainedin ascending orderbasedonthe keys.

SortedMapisgenericandisdeclared asshownhere:

interface SortedMap<K, V>**K**

specifies the type of keys,

Vspecifiesthetypeofvalues.

Several methods throwa **NoSuchElementException** when no items arein theinvokingmap. A**ClassCastException** is thrown when an object is incompatiblewith theelements in amap. A**NullPointerException**isthrownifanattemptismadetousea**null**objectwhen**null**isnot allowed in the map.

An**IllegalArgumentException**isthrownifaninvalidargumentisused.
Sorted maps allow very efficient manipulations of *submaps*

Toobtain asubmap, use**headMap( )**,**tailMap( )**,or**subMap( )**.

Togetthefirstkeyintheset,call**firstKey()**. To get

the last key, use **lastKey( )**.

NavigableMap Interface

The**NavigableMap**interfacewasadded byJavaSE 6.

Itextends**SortedMap**anddeclaresthebehaviorofamapthatsupportstheretrievalof entries based on the closest match to a given key or keys. **NavigableMap** is a generic interface that has this declaration:
interfaceNavigableMap<K,V>

Here, **K**specifiesthetypeofthekeys, and **V**specifiesthetypeofthevaluesassociatedwith the keys.

Severalmethodsthrowa**ClassCastException**whenanobjectisincompatiblewiththekeys in the map.

A**NullPointerException**isthrownifanattemptismadetousea**null**objectand**null**keys are not allowed in the set.

An**IllegalArgumentException**isthrownifaninvalidargumentisused. equal
to start.

Map.EntryInterface

The**Map.Entry**interfaceenablesyoutoworkwithamapentry.Recallthatthe**entrySet()** methoddeclared bythe**Map**interfacereturns a**Set**containingthemap entries.

Eachofthesesetelementsisa**Map.Entry**object.**Map.Entry**isgenericandisdeclaredlike this:
interfaceMap.Entry<K,V>Here,**K**specifiesthetypeofkeys,and**V**specifiesthetypeof values.
the methods declared by**Map.Entry**.

MapClasses

Severalclassesprovideimplementationsofthemapinterfaces.Theclassesthatcan beused for maps are summarized here:

HashMap:
The**HashMap**classextends**AbstractMap**andimplementsthe**Map**interface. Itusesahash table to store the map.

This allows the execution time of **get( )** and **put( )** to remain constant evenforlargesets.**HashMap**isagenericclassthathasthisdeclaration:

classHashMap<K,V>

Here,**K**specifies thetypeof keys,and**V**specifiesthe typeof values.

Thefollowingconstructorsare defined:
HashMap()
HashMap(Map<?extendsK,?extendsV>*m*)
HashMap(int *capacity*)
HashMap(int*capacity*,float *fillRatio*)
The first form constructs a default hash map. The second form initializes the hash map by usingtheelementsof*m*. Thethirdforminitializesthecapacityofthehash mapto *capacity*. The fourth form initializes both the capacity and fill ratio of the hash map by using its arguments.

Themeaningofcapacityandfillratioisthesameasfor **HashSet**,describedearlier.The default capacity is 16.

Thedefaultfill ratiois 0.75.
HashMapimplements**Map**andextends**AbstractMap**.Itdoesnotaddanymethodsof its own.

```
import
java.util.*;classHashM
apDemo{
publicstatic void main(Stringargs[]) {
HashMap<String,Double>hm=newHashMap<String,Double>();
hm.put("John Doe", new Double(3434.34));
hm.put("Tom Smith", new Double(123.22));
hm.put("Jane Baker", new Double(1378.00));
hm.put("Tod Hall", new Double(99.22));
hm.put("Ralph Smith", new Double(-19.08));
Set<Map.Entry<String,Double>>set=hm.entrySet();
for(Map.Entry<String, Double> me : set) {
System.out.print(me.getKey() + ": ");
System.out.println(me.getValue());
}
System.out.println();
double balance = hm.get("John Doe");
hm.put("John Doe", balance + 1000);
System.out.println("JohnDoe'snewbalance:"+
hm.get("John Doe"));
}
}
```

Outputfromthisprogramisshownhere(thepreciseordermayvary): Ralph
Smith: -19.08
Tom Smith: 123.22
John Doe: 3434.34
TodHall:99.22
JaneBaker:1378.0
JohnDoe’snewbalance:4434.34

Theprogram begins bycreatingahash mapand then adds themappingofnames to balances.Next,thecontentsofthemaparedisplayedbyusingaset-view, obtainedbycalling **entrySet( )**. The keys and values are displayed by calling the **getKey( )** and **getValue( )** methods thataredefinedby**Map.Entry**.Paycloseattentiontohowthedepositismadeintojohn Doe's account.The**put()**methodautomaticallyreplacesanypreexistingvaluethatisassociated with the specified key with the new value. Thus, after John Doe's account is updated, the hash map will still contain just one "John Doe" account.

TreeMap

The**TreeMap**classextends**AbstractMap**andimplementsthe**NavigableMap**interface. It creates maps stored in a tree structure.

A**TreeMap**providesanefficientmeansofstoringkey/valuepairsinsortedorderandallows rapid retrieval. You should note that, unlike a hash map, a tree map guarantees that its elements will be sorted in ascending key order.

TreeMapis agenericclassthat hasthisdeclaration:
classTreeMap<K,V>

Here,**K**specifiesthetypeofkeys,and **V**specifiesthetypeofvalues. The

following **TreeMap** constructors are defined:
TreeMap()
TreeMap(Comparator<? super K>*comp*)
TreeMap(Map<?extendsK,?extendsV>*m*)
TreeMap(SortedMap<K, ? extends V>*sm*)
The first form constructs an empty tree map that will be sorted by using the natural order of itskeys.Thesecondformconstructsanemptytree-basedmapthatwillbesortedbyusingthe **Comparator** *comp.* (Comparators are discussed later in this chapter.) The third form initializes
a tree map with the entries from *m,* which will be sorted by using the natural order of the keys.Thefourthforminitializesatreemapwiththeentriesfrom *sm*,whichwillbesortedin the same order as *sm.*
TreeMaphasnomethodsbeyondthosespecifiedbythe**NavigableMap**interfaceand the **AbstractMap** class.
Thefollowingprogramreworkstheprecedingexamplesothatituses **TreeMap**: import java.util.*;
classTreeMapDemo{
publicstatic void main(Stringargs[]) {
//Createatreemap.
TreeMap<String,Double>tm=newTreeMap<String,Double>();
//Put elements to the map.
tm.put("John Doe", new Double(3434.34));
tm.put("Tom Smith", new Double(123.22));
tm.put("JaneBaker",newDouble(1378.00));
tm.put("Tod Hall", new Double(99.22));
tm.put("RalphSmith",newDouble(-19.08));

```
//Geta setoftheentries.
Set<Map.Entry<String, Double>>set=tm.entrySet();
// Display the elements.
for(Map.Entry<String,Double>me:set){
System.out.print(me.getKey() + ": ");
System.out.println(me.getValue());
}
System.out.println();
double balance = tm.get("John Doe");
tm.put("John Doe", balance + 1000);
System.out.println("JohnDoe'snewbalance:"+
tm.get("John Doe"));
}
}
```

Thefollowingis theoutput from thisprogram:

```
JaneBaker:1378.0
John Doe: 3434.34
RalphSmith:-19.08
ToddHall:99.22
Tom Smith: 123.22
JohnDoe’scurrentbalance: 4434.34
```

TreeMapsortsthekeys.

However,inthiscase,theyaresortedbyfirstname
instead of last name.

Youcanalterthisbehaviorbyspecifyingacomparatorwhenthemap is
created, as described shortly.

LinkedHashMap

LinkedHashMapextends**HashMap**.

Itmaintainsalinked listoftheentries inthemap, in the 1orderinwhichtheywereinserted.Thisallowsinsertion-orderiterationoverthemap.That is, when iterating through a collection-view of a **LinkedHashMap**, the elements will be returned in the order in which they were inserted.

LinkedHashMapthatreturnsits elementsinthe orderinwhich theywere last accessed.

LinkedHashMapisagenericclassthathasthisdeclaration: class

LinkedHashMap<K, V>

Here,**K**specifiesthe typeofkeys, and**V**specifies thetypeof values.

LinkedHashMapdefinesthefollowingconstructors:

```
LinkedHashMap( )
LinkedHashMap(Map<?extendsK,?extendsV>m)
LinkedHashMap(int capacity)
```

LinkedHashMap(int *capacity*, float *fillRatio*)
LinkedHashMap(int*capacity*,float*fillRatio*,boolean*Order*) The
first form constructs a default **LinkedHashMap**.

Thesecondforminitializesthe **LinkedHashMap**withtheelementsfrom *m*.Thethirdform initializes the capacity. The fourth form initializes both capacityand fill ratio. The meaning of capacity and fill ratio are the same as for **HashMap**. The default capactiy is 16. The defaultratiois0.75.Thelastformallows youtospecifywhethertheelementswillbestored in the linked list by insertion order, or by order of last access.

IdentityHashMap

IdentityHashMapextends**AbstractMap**andimplementsthe**Map**interface.

Itissimilarto **HashMap** except that it usesreferenceequalitywhencomparingelements.
IdentityHashMapisagenericclassthathasthisdeclaration:
classIdentityHashMap<K,V>
Here,**K**specifiesthetypeofkey,and **V**specifiesthetypeofvalue.TheAPIdocumentation explicitly states that **IdentityHashMap** is not for general use.

TheEnumMap Class
EnumMapextends**AbstractMap**andimplements**Map**.Itisspecificallyforusewithkeys of an **enum** type. It is a generic class that has this declaration:
classEnumMap<K extendsEnum<K>,V>
Here, **K** specifies the type of key, and **V** specifies the type of value. Notice that **K** must extend**Enum<K>**,whichenforcestherequirementthatthekeysmustbeofan **enum**type.
EnumMap defines the following constructors:

EnumMap(Class<K>*kType*)
EnumMap(Map<K, ? extends V>*m*)
EnumMap(EnumMap<K,?extendsV>*em*)

Thefirstconstructorcreatesanempty**EnumMap** oftype*kType*.Thesecondcreatesan **EnumMap** map that contains the same entries as *m*. The third creates an **EnumMap** initialized with the values in *em*.

Comparatorinterface

Comparatorisa generic interfacethathasthisdeclaration:
interfaceComparator<T>
Here,**T**specifies thetypeof objectsbeingcompared.
The**Comparator**interfacedefinestwomethods:**compare( )**and**equals()**.The**compare()** method,shownhere,comparestwoelementsfororder:
intcompare(T*obj1*,T*obj2*)
*obj1*and*obj2* arethe objects to becompared.

Thismethod returnszero if theobjects areequal.
Itreturnsapositivevalue if *obj1* isgreaterthan*obj2*. Otherwise,anegative valueis returned.
ClassCastExceptionifthetypesoftheobjectsarenotcompatibleforcomparison. By

overriding **compare( )**, you can alter the way that objects are ordered.

Forexample,tosortinreverseorder, youcancreateacomparatorthatreversestheoutcome of a comparison. The **equals( )** method, shown here, tests whether an object equals the invoking comparator:
booleanequals(Object*obj*)
Here, *obj* is the object to be tested for equality. The method returns **true** if *obj* and the invokingobjectareboth **Comparator**objectsandusethesameordering.Otherwise,it returns **false**.

```
import java.util.*;
classMyCompimplementsComparator<String>{
public int compare(String a, String b) {
StringaStr,bStr;
aStr = a;
bStr =b;
return bStr.compareTo(aStr);
}
}
classCompDemo{
publicstatic void main(Stringargs[]) {
TreeSet<String>ts=newTreeSet<String>(newMyComp());
ts.add("C");
ts.add("A");
ts.add("B");
ts.add("E");
ts.add("F");
ts.add("D");
for(String element : ts)
System.out.print(element+"");
System.out.println();
}}
```

Output:
FE DC B A

TheCollectionAlgorithms:

CollectionsFrameworkdefinesseveralalgorithmsthatcanbeappliedtocollections and maps.

algorithmsaredefinedasstaticmethodswithinthe **Collections**class.

static<T> BooleanaddAll(Collection<?superT>c, T...elements)
Insertstheelementsspecified byelementsinto the collectionspecified byc.Returnstrueiftheelementswereaddedandfalseotherwise.

static<T>Queue<T>asLifoQueue(Deque<T>c)
Returns a last-in, first-out view of c.

static<T>intbinarySearch(List<?extendsT>list,Tvalue,Comparator<?superT>c)
Searchesforvalueinlistorderedaccordingtoc. Returnsthepositionofvalue in list, or a negative value if value is not found.

static<T>intbinarySearch(List<?extendsComparable<?superT>>list,Tvalue)

Searchesforvalueinlist.Thelistmustbesorted.Returnsthepositionofvalue in list, or a negative value if value is not found.

static<E>Collection<E>checkedCollection(Collection<E>c,Class<E>t)
Returnsarun-timetype-safeviewofacollection. Anattempttoinsertan incompatible element will cause a ClassCastException.

static<E>List<E> checkedList(List<E>c,Class<E>t)
Returnsarun-timetype-safeviewofaList.Anattempttoinsertan incompatible element will cause a ClassCastException.

static<K,V>Map<K,V>checkedMap(Map<K,V>c,Class<K>keyT,Class<V>valueT)
Returnsarun-time type-safeviewof aMap.
Anattempttoinsertanincompatibleelementwillcausea ClassCastException.

static<E>List<E> checkedSet(Set<E>c,Class<E>t)
Returnsarun-timetype-safeviewofaSet. An attempttoinsertanincompatibleelementwillcauseaClassCastException.

staticintfrequency(Collection<?>c,Objectobj)
Countsthenumberofoccurrencesofobjincand returnstheresult. static int indexOfSubList(List<?> list, List<?> subList)
Searcheslistforthe firstoccurrenceofsubList.
Returnstheindexofthefirstmatch,or–1ifnomatchisfound. static int lastIndexOfSubList(List<?> list, List<?> subList)
SearcheslistforthelastoccurrenceofsubList.
Returnstheindex ofthelastmatch,or–1ifno match is found.

```
import java.util.*;
classAlgorithmsDemo{
public static void main(String args[]) {
LinkedList<Integer>ll=newLinkedList<Integer>();
ll.add(-8);
ll.add(20);
ll.add(-20);
ll.add(8);
Comparator<Integer>r=Collections.reverseOrder();
Collections.sort(ll, r);
System.out.print("Listsortedinreverse:"); for(int
i : ll)
System.out.print(i+ "");
System.out.println();
Collections.shuffle(ll);
System.out.print("Listshuffled:");
for(int i : ll)
System.out.print(i+"");
System.out.println();
System.out.println("Minimum:"+Collections.min(ll));
```

```
System.out.println("Maximum:"+ Collections.max(ll));
}
}
```

Output:
Listsortedinreverse:208 -8-20
Listshuffled:20-208-8
Minimum: -20
Maximum:20
Noticethat**min()**and**max()**operateonthelistafterithasbeenshuffled.Neitherrequires a sorted list for its operation.

WhyGeneric Collections?
Asmentionedatthestartofthischapter,theentireCollectionsFrameworkwasrefittedfor generics when JDK 5 was released.

Furthermore,theCollectionsFrameworkisarguablythesinglemostimportantuseof generics in the Java API.

ThereasonforthisisthatgenericsaddtypesafetytotheCollectionsFramework.Before moving on, it is worth taking some time to examine in detail the significance of this improvement.

```
importjava.util.*;
class OldStyle {
publicstaticvoidmain(Stringargs[]){
ArrayList list = new ArrayList();
list.add("one");
list.add("two");
list.add("three");
list.add("four");
Iteratoritr=list.iterator();
while(itr.hasNext()) {
String str = (String) itr.next(); // explicit cast needed here.
System.out.println(str+"is"+str.length()+"charslong.");
}
}
}
```

Priortogenerics,allcollectionsstoredreferencesoftype**Object**. This

allowed any type of reference to be stored in the collection.

Theprecedingprogramusesthisfeaturetostore referencestoobjectsoftype **String**in**list**, butanytypeofreferencecouldhavebeenstored. Unfortunately,thefactthatapre-generics collection stored **Object** references could easily lead to errors.

First,itrequiredthatyou,ratherthanthecompiler,ensurethatonlyobjectsof thepropertypebestoredinaspecificcollection.Forexample,intheprecedingexample, **list** is clearly intended to store **String**s, but there is nothing that actually prevents another typeof reference from being added to the collection.

Forexample,thecompilerwillfindnothingwrongwiththislineofcode:
list.add(new Integer(100));

Because**list**stores**Object**references,itcanstoreareferenceto**Integer**aswellasitcan store a reference to **String**.
However,if youintended **list**toholdonlystrings,thentheprecedingstatementwouldcorrupt the collection. Again, the compiler had no way to know that the preceding statement is invalid.

Thesecondproblemwithpre-genericscollectionsisthatwhenyouretrieveareference from the collection, you must manually cast that reference into the proper type.
Thisiswhytheprecedingprogramcaststhereferencereturnedby**next()**into**String**.Prior togenerics,collectionssimplystored**Object**references.Thus,thecastwasnecessarywhen retrieving
objectsfromacollection.

Aside from the inconvenience of always having to cast a retrieved reference into its proper type, this lack of type safety often led to a rather serious, but surprisingly easy-to-create, error.Because**Object** canbecastintoanytypeofobject,itwaspossibleto castareference obtainedfromacollectionintothe*wrongtype*.Forexample,ifthefollowingstatementwere addedtotheprecedingexample,itwouldstillcompilewithouterror,but generatearun-time exception when executed:

Integeri=(Integer)itr.next();

• Ensuresthatonlyreferencestoobjectsofthepropertypecanactuallybestoredin a collection. Thus, a collection will always contain references of a known type.
• Eliminatestheneedtocastareferenceretrievedfromacollection.Instead,areference retrieved from a collection is automatically cast into the proper type. This prevents run-timeerrorsduetoinvalidcastsand avoidsanentirecategoryoferrors.

TheLegacyClassesandInterfaces

As explained at the start of this chapter, early versions of **java.util** did not include the CollectionsFramework.Instead,itdefinedseveralclassesandaninterfacethatprovidedan ad hoc method of storing objects.

Whencollectionswereadded(byJ2SE1.2),severaloftheoriginalclasseswerereengineered to support the collection interfaces.

Thus,theyarefullycompatiblewiththeframework.Whilenoclasseshaveactuallybeen deprecated, one has been rendered obsolete.

Of course, where a collection duplicates the functionalityof a legacyclass,you will usually wanttousethecollectionfornewcode.Ingeneral,thelegacyclassesare supportedbecause there is still code that uses them.

Oneotherpoint:noneofthecollectionclassesaresynchronized,butallthelegacyclasses are synchronized.

Thisdistinctionmaybeimportantinsomesituations.Ofcourse,youcan easilysynchronizecollections,too, byusingoneofthealgorithmsprovidedby**Collections**.

Thelegacyclassesdefinedby**java.util**areshownhere: Dictionary
Hashtable
Properties
Stack
Vector
Thereis onelegacyinterface called**Enumeration**.

Thefollowingsectionsexamine**Enumeration**andeachofthelegacyclasses,inturn. The Enumeration Interface
The**Enumeration**interfacedefinesthemethodsbywhich youcan *enumerate*(obtainoneat a time) the elements in a collection of objects. This legacyinterface has been superseded by **Iterator**.
interfaceEnumeration<E>
where**E** specifiesthe typeof elementbeing enumerated.

Vector

Vectorimplementsadynamicarray.Itissimilarto **ArrayList**,butwithtwodifferences: **Vector** is synchronized, and it contains many legacy methods that are not part of the Collections.
Vectorisdeclaredlike this:
classVector<E>
Here,**E**specifiesthetypeofelementthatwillbestored.

Herearethe**Vector**constructors:
Vector( )
Vector(int*size*)
Vector(int *size*, int *incr*)
Vector(Collection<?extendsE>*c*)

- Thefirstformcreates adefaultvector, whichhasan initialsizeof 10.
- Thesecond form createsavectorwhoseinitial capacityis specified by*size.*
- Thethirdformcreatesavectorwhoseinitialcapacityisspecifiedby*size*andwhose increment is specified by *incr*.

Theincrementspecifiesthenumberofelementstoallocateeachtimethatavectorisresized upward.
Thefourthformcreatesavectorthatcontainsthe elementsofcollection*c.*

Stack

Stackisasubclassof**Vector**thatimplementsastandardlast-in,first-outstack.**Stack**only defines the default constructor, which creates an empty stack. With the release of JDK 5, **Stack** was retrofitted for generics and is declared as shown here:

classStack<E>

Here,**E**specifiesthe typeofelement storedinthestack.
Stack includes all themethods defined by**Vector.**

Dictionary

Dictionaryisanabstractclassthatrepresentsakey/valuestoragerepositoryandoperates much like **Map**.
Givenakeyandvalue,you canstorethe valueina**Dictionary**object. Once thevalueisstored, youcanretrieveitbyusingits key.Thus,likeamap,adictionarycanbe thought of as a list of key/value pairs.
Although not currentlydeprecated,**Dictionary** is classifiedasobsolete,becauseitisfullysupersededby**Map**.However,**Dictionary**isstillin use and thus is fully discussed here.

classDictionary<K, V>

Here,**K**specifiesthetypeofkeys,and **V**specifiesthetypeofvalues.Theabstractmethods defined by **Dictionary** are listed in Table 17-17.

Hashtable

Hashtablewaspartoftheoriginal**java.util**andisaconcreteimplementationofa Dictionary.
HashMap,**Hashtable**storeskey/valuepairsina hashtable.However,neitherkeys norvaluescanbe**null**.Whenusinga**Hashtable**, youspecifyanobjectthatisusedasakey, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
Hashtablewas madegenericbyJDK 5.
Itisdeclaredlikethis: classHashtable<K,V>

Hashtable( )
Hashtable(int*size*)
Hashtable(int *size*, float *fillRatio*)
Hashtable(Map<?extendsK,?extendsV>*m*)

Thefirstversion isthedefault constructor.

Thesecondversioncreatesahashtablethathas an initial size specified by *size*.

The third version creates a hash table that has an initial size specified by*size* and a fill ratio specifiedby*fillRatio*.Thisratiomustbebetween 0.0and1.0,anditdetermineshowfullthe hash table can be before it is resized upward. Specifically, when the number of elements is greater than the capacity of the hashtable multiplied by its fill ratio, the hash table is expanded. If you do not specify a fill ratio,
then 0.75 is used.

Finally,thefourthversion createsahashtablethatisinitializedwith the elementsin *m*.Thecapacityof thehash table is setto twicethe number of elements in*m*. Thedefaultloadfactorof 0.75is used.

Properties

Propertiesisasubclass of**Hashtable**. Itisusedtomaintainlistsofvaluesinwhichthe key is a **String** and the value is also a **String**.

The**Properties**classisusedbymanyotherJavaclasses.Forexample,itisthetypeofobject returned by **System.getProperties( )** when obtaining environmental values.

Althoughthe **Properties**class,itself,isnotgeneric,several ofitsmethodsare.
Propertiesdefinesthe followinginstance variable:

Propertiesdefaults;

Thisvariableholdsadefaultpropertylist associatedwith a**Properties** object.**Properties** definestheseconstructors:
Properties( )
Properties(Properties*propDefault*)

Module 2

Syllabus-StringHandling:

The String Constructors, String Length, Special String Operations, String Literals, String Concatenation, String Concatenation with Other Data Types, String Conversion and toString( ) Character Extraction, charAt( ), getChars( ), getBytes( ) toCharArray(), String Comparison,equals()andequalsIgnoreCase(),regionMatches()startsWith()andendsWith( ),equals()Versus==,compareTo()SearchingStrings,ModifyingaString,substring(), concat( ), replace( ), trim( ), Data Conversion Using valueOf( ), Changing the Case of Characters Within a String, Additional String Methods, StringBuffer , StringBuffer Constructors, length( ) and capacity( ), ensureCapacity( ), setLength( ), charAt( ) and setCharAt( ), getChars( ),append( ), insert( ), reverse( ), delete( ) and deleteCharAt( ), replace( ),substring( ),Additional StringBuffer Methods, StringBuilder.

1. **WhatarethedifferenttypesofStringConstructorsavailableinJava?**

   TheStringclasssupportsseveralconstructors.

   a. TocreateanemptyString
      thedefaultconstructor isused.

   Ex:Strings =new String();
   willcreate an instanceof Stringwithno charactersin it.

   b. TocreateaStringinitializedbyanarrayofcharacters,usetheconstructor shown here:
      String(charchars[])
      ex: char chars[] = { 'a', 'b', 'c' };
      Strings=newString(chars);
      Thisconstructorinitializeswiththestring“abc”.

   c. Tospecifyasubrangeofacharacterarrayasaninitializerusingthefollowing constructor:
      String(charchars[],intstartIndex,intnumChars)

      Here,startIndexspecifiestheindexatwhichthesubrangebegins,and numChars specifies the number of characters to use. Here is an example:
      charchars[]={'a','b','c','d','e','f'}; String s
      = new String(chars, 2, 3);
      Thisinitializesswiththecharacterscde.

   d. ToconstructaStringobjectthatcontainsthesamecharactersequenceas another String object using this constructor:
      String(StringstrObj)
      Here,strObj isaStringobject.

      classMakeString
      {publicstatic void main(Stringargs[])
      {charc[] ={'J','a', 'v','a'};
      Strings1 =new String(c);

```
 Strings2=newString(s1);
System.out.println(s1);
System.out.println(s2);
}
}
```

Theoutputfromthisprogramisas follows:

```
Java
Java
```

As youcansee,s1ands2containthesame string.

e. ToConstruct stringusing bytearray:

Even though Java's char type uses 16 bits to represent the basic Unicode characterset,thetypicalformatforstringsontheInternetusesarraysof 8-bitbytes constructed from the ASCII character set.

Because8-bitASCIIstringsarecommon,theStringclassprovides constructors that initialize a string when given a byte array.

Ex:String(byteasciiChars[ ])
String(byteasciiChars[],intstartIndex,intnumChars)

Thefollowingprogramillustratestheseconstructors:

```
class SubStringCons
{publicstatic void main(Stringargs[])
{
byte ascii[] ={65,66, 67,68,69,70 };
Strings1=newString(ascii);
System.out.println(s1);
Strings2=newString(ascii,2,3);
System.out.println(s2);
}
}
```

Thisprogramgeneratesthefollowingoutput:

```
ABCDEF
CDE
```

f. ToconstructaStringfromaStringBufferbyusingtheconstructorshown here:
Ex:String(StringBuffer strBufObj)

g. ConstructingstringusingUnicodecharactersetandisshownhere:
String(int codePoints[ ], int startIndex, int numChars)

codePointsisan arraythatcontains Unicode codepoints.

h. ConstructingstringthatsupportsthenewStringBuilderclass. Ex :
String(StringBuilder strBuildObj)

Note:
Stringcanbeconstructedbyusingstringliterals.
String s1=”Hello World”

Stringconcatenation canbedoneusing+operator. With otherdata typealso.

String Length

1. Thelengthofastringisthenumberofcharactersthatitcontains.Toobtain this value, call the length( ) method,
2. Syntax:
   intlength()
3. Example
   charchars[] ={'a','b','c'};Strings=new String(chars);
   System.out.println(s.length());//3

toString()

1. EveryclassimplementstoString()becauseitisdefinedbyObject. However, the default Implementation of toString() is sufficient.
2. FormostimportantclassesthatyoucreateWillwanttooverridetoString() and provide your own string representations.
   StringtoString()
3. To implement toString( ), simply return a String object that contains the human-readablestringthatappropriatelydescribesanobjectof yourclass.
4. ByoverridingtoString() forclasses that you create, you allow them to be fullyintegratedintoJava’sprogrammingenvironment.Forexample,they can be used in print() and println() statements and in concatenation expressions.

5. Thefollowingprogramdemonstratesthisbyoverriding
   toString( ) for the Box class:

```
classBox
{
doublewidth;doubleheight;doubledepth;
Box(double w, double h, double d)
{width =w; height=h; depth=d; }

publicStringtoString()
{ return"Dimensions are"+width +"by"+depth +"by"+height +".";}
}

classtoStringDemo{
publicstatic void main(Stringargs[])
{
Boxb=newBox(10,12,14);
String s = "Box b: " + b;
```

```
System.out.println(b);
System.out.println(s);
}
}
```

Theoutput of this program is shown here:
Dimensions are 10.0by14.0by 12.0
Box b: Dimensionsare10.0 by14.0 by 12.0

CharacterExtraction

The String class provides a number of ways in which characters can be extracted from a String object.String object can not be indexed as if they were a character array, many of the String methods employ an index (or offset)intothestringfortheiroperation.Likearrays,thestringindexesbegin at zero.

A. charAt()

1. description:

   ToextractasinglecharacterfromaString, youcanreferdirectlytoan individual character via the charAt( ) method.

2. Syntax

   charcharAt(intwhere)

Here,whereistheindexofthecharacterthat youwanttoobtain. charAt( ) returns the character at the specified location.

3. example,

```
char ch;
 ch = "abc".charAt(1);
```

assignsthevalue“b”toch.

B. getChars()

1. toextractmorethanonecharacteratatime, youcanusethegetChars() method.
2. Syntax

   voidgetChars(intsourceStart,intsourceEnd,chartarget[],int targetStart)

   Here, sourceStart specifies the index of the beginning of the substring, sourceEnd specifies an index that is one past the end of the desired The array thatwillreceivethecharactersisspecifiedbytarget.Theindexwithintargetat which the substring will be copied is passed in targetStart.

3.
```
classgetCharsDemo{
publicstatic void main(Stringargs[])
{Strings="ThisisademoofthegetCharsmethod."; int
start = 10;
 intend =14;
 charbuf[]=newchar[end-start];
  s.getChars(start, end, buf, 0);
```

```
        System.out.println(buf);
    }
}
```

Hereis theoutputof thisprogram:

demo

C. getBytes()

1. ThismethodiscalledgetBytes(),anditusesthedefaultcharacter-to-byte conversions provided by the platform.

   Syntax:

   byte[]getBytes()

   Otherformsof getBytes( )arealsoavailable.

2. getBytes( ) is most useful when you are exporting a String value into an environmentthatdoesnotsupport16-bitUnicodecharacters.Forexample, most Internet protocols and text file formats use 8-bit ASCII for all text interchange.

D. toCharArray()

If youwanttoconvertallthecharactersinaStringobjectintoacharacter array, the easiest way is to call toCharArray( ).

Itreturnsanarrayofcharactersfortheentirestring. It has

this general form:

char[]toCharArray()

2. String Comparison:

TheStringclassincludesseveralmethodsthatcomparestringsor substrings within strings.

equals()

Tocomparetwostringsforequality,useequals(). It has this general form:

booleanequals(Objectstr)

Here, str is the Stringobject beingcompared with the invoking Stringobject. Itreturnstrueifthestringscontainthesamecharactersinthesameorder,and false otherwise. The comparison is case-sensitive.

A. equalsIgnoreCase()

To perform a comparison that ignores case differences, call equalsIgnoreCase().WhenitcomparestwostringsitconsidersA-Ztobethe same as a-z.

Ithasthisgeneralform:

booleanequalsIgnoreCase(String str)

Here, str is the String object being compared with the invoking String object. It,too,returnstrueifthestringscontainthesamecharactersinthesameorder, and false otherwise.

```
//Demonstrateequals()andequalsIgnoreCase().
class equalsDemo {
 publicstatic voidmain(Stringargs[]) {
```

```
String s1 = "Hello";
String s2 = "Hello";
Strings3="Good-bye";
String s4 = "HELLO";
System.out.println(s1+"equals"+s2+"->"+s1.equals(s2));
System.out.println(s1+"equals"+s3+"->"+s1.equals(s3));
System.out.println(s1+"equals"+s4+"->"+s1.equals(s4)); System.out.println(s1
+ " equalsIgnoreCase " + s4 + " ->" + s1.equalsIgnoreCase(s4)); } }
```

Theoutput fromtheprogramis shown here:
Hello equals Hello -> true
HelloequalsGood-bye->false
Hello equals HELLO -> false
HelloequalsIgnoreCaseHELLO->true

B. regionMatches()

1. TheregionMatches()methodcomparesaspecificregioninsideastringwith another specific region in another string. There is an overloaded form that allows you to ignore case in such comparisons.
2. Syntax:
   booleanregionMatches(intstartIndex,Stringstr2,intstr2StartIndex,int numChars)

   booleanregionMatches(booleanignoreCase,intstartIndex,Stringstr2, int str2StartIndex, int numChars)

3. Forbothversions,startIndexspecifiestheindexatwhichthe region begins within the invoking String object.

The String being compared is specified by str2. The index at which the comparisonwillstartwithinstr2isspecifiedbystr2StartIndex.Thelengthof the substring being compared is passed in numChars.

4. Inthesecondversion,ifignoreCaseistrue,thecaseofthe characters is ignored. Otherwise, case is significant.

C. startsWith()andendsWith()

1. ThestartsWith()methoddetermineswhetheragivenStringbeginswitha specified string.
2. endsWith()determineswhethertheStringinquestionendswitha specified string.
3. Syntax

```
booleanstartsWith(Stringstr)
boolean endsWith(String str)
```

Here,str isthe Stringbeingtested.
Ifthestringmatches,trueisreturned.
Otherwise, false is returned.
Forexample,
"Foobar".endsWith("bar")

"Foobar".startsWith("Foo")
are both true.

4. AsecondformofstartsWith(),shownhere,letsyouspecifyastarting point:
booleanstartsWith(Stringstr,intstartIndex)

Here,startIndexspecifies theindexintotheinvokingstringatwhichpointthe search will begin. For example,
"Foobar".startsWith("bar",3)
returns true.

D. **equals()Versus==**

Itisimportanttounderstandthattheequals()methodandthe==operator perform two different operations.
theequals()methodcomparesthe charactersinsideaStringobject.
The==operatorcomparestwoobjectreferencestoseewhethertheyreferto the same instance.

```
classEqualsNotEqualTo {
publicstaticvoidmain(Stringargs[]){ String
s1 = "Hello";
Strings2 =new String(s1);
System.out.println(s1+"equals"+s2+"->"+s1.equals(s2)); System.out.println(s1
+ " == " + s2 + " ->" + (s1 == s2));
}
}
```

E. **compareTo()**

1. Sortingapplications, youneedtoknowwhichislessthan,equalto,or greater than the next.
2. A stringis less than another if it comes before the otherin dictionary order. A string is greater than another if it comes after the other in dictionaryorder.TheStringmethodcompareTo()servesthispurpose.
3. Ithasthisgeneralform:
intcompareTo(String str)
Here,stristheStringbeingcomparedwiththeinvokingString.The result of the comparison is returned and is interpreted,
4. Lessthanzerowheninvokingstringislessthanstr.
5. Greaterthan zerowhen invokingstringisgreaterthan str.
6. ZeroThetwostringsare equal.

```
//AbubblesortforStrings.
class SortString
{staticStringarr[]={"Now", "is","the", "time","for", "all", "good","men",
"to","come","to","the","aid","of","their","country"};
 public static void main(String args[])
 {for(intj =0;j <arr.length; j++)
{for(inti =j +1;i <arr.length; i++)
 {if(arr[i].compareTo(arr[j])<0)
 {Stringt=arr[j];
arr[j] = arr[i];
arr[i] = t;
}
} System.out.println(arr[j]);
}
}
}
```

Theoutput of this program is the list of words:

Nowaidallcomecountryforgoodismenofthethetheirtimetoto As you can see

7. Ignore case differences when comparing two strings, use compareToIgnoreCase(),Thismethodreturnsthesameresultsas compareTo(), except that case differences are ignored.

5. SearchingString

A.indexOf()andlastIndexOf()

1. indexOf()Searchesforthefirstoccurrenceofacharacteror substring.

2. lastIndexOf()Searchesforthelastoccurrenceofacharacteror substring.

3. Thesetwomethodsareoverloadedinseveraldifferent ways

4. returntheindexatwhichthecharacterorsubstringwasfound,or –1 on failure.

5. Tosearchforthe firstoccurrenceof acharacter,intindexOf(int ch)
6. Tosearchforthelastoccurrence ofa character, intlastIndexOf(intch)Here,chisthe characterbeingsought

7. To search for the first or last occurrence of a substring, use int indexOf(Stringstr)intlastIndexOf(Stringstr)Here,strspecifies the substring.

8. Youcanspecifyastartingpointforthesearchusingtheseforms: int indexOf(int ch, int startIndex)
   intlastIndexOf(intch,intstartIndex)

9. intindexOf(Stringstr,intstartIndex)intlastIndexOf(Stringstr,int startIndex) Here, startIndex specifies the index at which point the search begins.
10. For indexOf( ), the search runs from startIndex to the end of the string.ForlastIndexOf(),thesearchrunsfromstartIndextozero. The following example shows how to use the various index methods to search inside of Strings:

```
//DemonstrateindexOf()andlastIndexOf().
class indexOfDemo {
 publicstatic void main(Stringargs[])
{Strings="Nowisthetimeforall goodmen"+"tocometotheaid of their country.";
System.out.println(s);
System.out.println("indexOf(t)="+s.indexOf('t'));
System.out.println("lastIndexOf(t) = " + s.lastIndexOf('t'));
System.out.println("indexOf(the) = " + s.indexOf("the"));
System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));
System.out.println("indexOf(t, 10) = " + s.indexOf('t', 10));
System.out.println("lastIndexOf(t,60)= "+s.lastIndexOf('t',60));
System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));
System.out.println("lastIndexOf(the,60)="+s.lastIndexOf("the", 60));
}
}
```

Output

```
Nowisthetimeforallgoodmentocometotheaidoftheircountry. indexOf(t) = 7
lastIndexOf(t)=65
 indexOf(the)=7
lastIndexOf(the)=55
indexOf(t,10)=11
lastIndexOf(t,60)=55
indexOf(the,10)=44
 lastIndexOf(the,60)=55
```

6. Modifyinga String

String objects are immutable, whenever you want to modify a String, youmusteithercopyitintoaStringBufferorStringBuilder,oruseone of the following String methods, which will construct a new copy of the string with your modifications complete.

A. Substring()

1. Youcanextractasubstringusingsubstring().Ithastwoforms.The first is String substring(int startIndex)
2. Here, startIndex specifies the index at which thesubstringwill begin. Thisformreturnsacopyofthesubstringthatbeginsatstartindexand runs to the end of the invoking string.
3. Thesecondformofsubstring()allows youtospecifybothbthe beginning and ending index of the substring:
   Stringsubstring(int startIndex, int endIndex)
   Here,startIndexspecifiesthebeginningindex,andendIndexspecifies the stopping point.
4. The string returned contains all the characters from the beginning index, up to, but not including, the ending index. The following programusessubstring()toreplaceallinstancesofonesubstringwith another within a string:

```
//Substringreplacement.
class StringReplace {
public static void main(String args[]) {
Stringorg="Thisisatest.Thisis,too.";
String search = "is";
Stringsub="was";
String result = "";
int i;
do{
System.out.println(org);
i = org.indexOf(search);
 if(i!=-1){result=org.substring(0,i); result
= result + sub;
result=result+org.substring(i+search.length()); org
 = result;
}}while(i!=-1);
}
}
```

Theoutputfromthisprogramisshownhere: This
is a test. This is, too.
Thwas is a test. This is, too.
Thwas was a test. This is, too.
Thwas was a test. Thwas is, too.
Thwaswasatest.Thwaswas,too.

B. concat()

1. concatenatetwostringsusingconcat()

String concat(String str)

2. Thismethodcreatesanewobjectthatcontainstheinvokingstring with the contents of str appended to the end.
3. concat()performsthesamefunctionas+.
4. Strings1 ="one";
   Strings2 =s1.concat("two");

C. replace()

1. Thereplace( ) methodhastwo forms.
2. Thefirstreplacesalloccurrencesofonecharacterinthe invoking string with another character.

Syntax:

Stringreplace(charoriginal,char replacement)

Here,originalspecifiesthecharactertobereplacedbythecharacter specified by replacement. The resulting string is returned.

Example

Strings="Hello".replace('l','w');

puts the string “Hewwo” into s.

Thesecondformofreplace()replacesonecharactersequencewith another. It has this general form:

Stringreplace(CharSequenceoriginal,CharSequence replacement)

D. trim()

Thetrim()methodreturnsacopyoftheinvokingstringfromwhich any leading and trailing whitespace has been removed.

Syntax:

Stringtrim()

Example:

String s = "Hello World

".trim();This

putsthestring“HelloWorld”intos.

Thetrim()methodisquiteusefulwhenyou processusercommands.

```
//Usingtrim()toprocesscommands. import
java.io.*;
classUseTrim
{publicstaticvoidmain(Stringargs[])throwsIOException{
BufferedReader br = new BufferedReader(new
nputStreamReader(System.in));
Stringstr;
System.out.println("Enter'stop'toquit.");
System.out.println("Enter State: ");
do { str =br.readLine();
```

```
 str = str.trim();
if(str.equals("Illinois"))
System.out.println("Capital is Springfield.");else
if(str.equals("Missouri"))
System.out.println("Capital is Jefferson City.");
else if(str.equals("California"))
System.out.println("Capital is Sacramento.");
else if(str.equals("Washington"))
System.out.println("Capital is Olympia."); // ... }
while(!str.equals("stop"));
}
}
```

5. DataConversion

1. ThevalueOf()methodconvertsdatafromitsinternalformatinto a human-readable form.
2. It is a static method that is overloaded within String for all of Java'sbuilt-intypessothateachtypecanbeconvertedproperly into a string.

3. valueOf()isalsooverloadedfortypeObject,soanobjectofany class type you create can also be used as an argument

Syntax:

```
static String valueOf(double num)
static String valueOf(long num)
static String valueOf(Object ob)
staticStringvalueOf(charchars[])
```

4. valueOf()iscalledwhenastringrepresentationofsomeothertype of data is needed. example, during concatenation operations.

5. Anyobjectthat youpasstovalueOf()willreturntheresultofa call to the object's toString( ) method.

6. ThereisaspecialversionofvalueOf()thatallows youtospecifya subset of a char array.

Syntax:

```
staticStringvalueOf(charchars[ ],intstartIndex,int numChars)
```

7. Here,charsisthearraythatholdsthecharacters,startIndexisthe index into the array of characters at which the desired substring begins, and numChars specifies the length of the substring.

6. ChangingCaseofCharacters

A. toLowerCase()

1. convertsallthecharacters inastringfromuppercaseto lowercase.

2. ThismethodreturnaStringobjectthatcontainsthelowercase equivalent of the invoking String.

3. Nonalphabeticalcharacters,suchas digits,are unaffected.

Syntax

StringtoLowerCase()

B. toUpperCase()

1. convertsallthecharactersinastringfromlowercaseto uppercase.
2. ThismethodreturnaStringobjectthatcontainstheuppercase equivalent of the invoking String.

3. Nonalphabeticalcharacters,suchasdigits,are unaffected.

Syntax

StringtoUpperCase( )

```
classChangeCase{
 publicstaticvoidmain(Stringargs[]){
String s = "This is a test.";
System.out.println("Original: " + s);
String upper = s.toUpperCase();
String lower = s.toLowerCase();
System.out.println("Uppercase:"+upper);
System.out.println("Lowercase:"+lower);
}
}
```

Output:

Original: This is a test.

Uppercase:THISISATEST.

Lowercase: this is a test.

StringBuffer

StringBufferisapeerclassofStringthatprovidesmuchofthefunctionalityofstrings.As you know, String represents fixed-length, immutable character sequences.

StringBufferrepresents growableandwriteablecharacter sequences.

StringBuffermayhavecharactersandsubstringsinsertedinthemiddleorappendedtothe end.

StringBufferwillautomaticallygrowtomakeroomforsuchadditionsandoftenhasmore characters pre allocated than are actually needed, to allow room for growth.

StringBufferConstructors

StringBufferdefinesthesefourconstructors:

StringBuffer( )

StringBuffer(int size)

StringBuffer(String str)

StringBuffer(CharSequencechars)

a. Thedefaultconstructor(theonewithnoparameters)reservesroomfor16characters without reallocation.
b. Thesecondversionacceptsanintegerargumentthatexplicitlysetsthesizeofthe buffer.
c. The third version accepts a String argument that sets the initial contents of the StringBufferobjectandreservesroomfor16morecharacterswithoutreallocation.

d. StringBufferallocatesroomfor16additionalcharacterswhennospecificbuffer length is requested, because reallocation is a costly process in terms of time.

A. length()andcapacity()

a. ThecurrentlengthofaStringBuffercanbefoundviathelength()method,whilethe total allocated capacity can be found through the capacity( ) method.

Syntax

int length( )

intcapacity()

b. Example:

```
classStringBufferDemo
{
  publicstatic void main(Stringargs[])
  {
     StringBuffer sb = new StringBuffer("Hello");
    System.out.println("buffer = " + sb);
    System.out.println("length = " + sb.length());
    System.out.println("capacity="+sb.capacity());
}
```

}

Output

buffer=Hello

length = 5

capacity= 21

B. ensureCapacity()

a. If you want to pre allocate room for a certain number of characters after a StringBufferhasbeenconstructed, youcanuseensureCapacity()tosetthesizeofthe buffer.
b. Thisisusefulifyouknowinadvancethat youwillbeappendingalargenumberof small strings to a StringBuffer.
   Syntax

   voidensureCapacity(int capacity)

   Here,capacityspecifiesthe sizeofthe buffer.

C. setLength()

a. TosetthelengthofthebufferwithinaStringBufferobject,

Syntax:

voidsetLength(intlen)

Here,lenspecifiesthelengthof thebuffer. Thisvaluemustbenonnegative.

Whenyouincreasethesizeofthebuffer,nullcharactersareaddedtotheendoftheexisting buffer.

If youcallsetLength()withavaluelessthanthecurrentvaluereturnedbylength(),thenthe characters stored beyond the new length will be lost.

D. charAt()andsetCharAt()

a. The value of a single character can be obtained from a StringBuffer via the charAt()method.YoucansetthevalueofacharacterwithinaStringBufferusing setCharAt( ).
b. Syntax

   charcharAt(intwhere)

   voidsetCharAt(intwhere,charch)

c. ForcharAt( ),wherespecifiestheindexofthecharacterbeingobtained.
d. ForsetCharAt(),wherespecifiestheindexofthecharacterbeingset,andchspecifies the new value of that character.

```
//DemonstratecharAt()andsetCharAt().
 class setCharAtDemo {
publicstatic void main(Stringargs[])
 { StringBuffer sb = new StringBuffer("Hello");
System.out.println("buffer before = " + sb);
System.out.println("charAt(1) before = " + sb.charAt(1));
sb.setCharAt(1, 'i');
sb.setLength(2);
System.out.println("bufferafter="+sb);
System.out.println("charAt(1)after="+sb.charAt(1));} }
```

Output

bufferbefore=Hello

charAt(1) before = e

buffer after = Hi

charAt(1) after = i

E. getChars()

a. TocopyasubstringofaStringBufferintoanarray,usethe getChars()method. Syntax

Syntax

voidgetChars(intsourceStart,intsourceEnd,char target[],inttargetStart)

Here,sourceStartspecifiestheindexofthebeginningofthesubstring,andsourceEnd specifies an index that is one past the end of the desired substring.

b. Thismeansthatthesubstringcontainsthecharactersfrom sourceStartthrough sourceEnd–1.

c. Thearraythat will receivethecharacters is specified bytarget.

Theindexwithin targetwhichthesubstringwillbecopiedis passed in targetStart.

d. Caremustbetakentoassurethatthetargetarrayislargeenoughtoholdthenumber of characters in the specified substring.

F. append()

1. Theappend()methodconcatenatesthestringrepresentationofanyothertypeofdata to the end of the invoking StringBuffer object. It has several overloaded versions. Here are a few of its forms:

StringBufferappend(Stringstr)

StringBuffer append(int num)

StringBufferappend(Objectobj)

2. TheresultisappendedtothecurrentStringBufferobject.
3. Thebuffer itselfis returned byeach version ofappend( ).
4. Thisallowssubsequentcallstobechainedtogether,asshowninthefollowing example:

```
classappendDemo {
 publicstatic void main(Stringargs[])
 { Strings; int a = 42;
  StringBuffersb=newStringBuffer(40);
  s=sb.append("a=").append(a).append("!").toString();
  System.out.println(s);
 }
}
```

Output

a=42!

G. **insert()**

1. Theinsert()method inserts one stringin toanother.
2. Itisoverloadedtoacceptvaluesofallthesimpletypes,plusStrings,Objects,and CharSequences.
3. Likeappend(),itcallsString.valueOf()toobtainthestringrepresentationofthevalue it is called with.
4. Thisstringisthen insertedinto theinvokingStringBuffer object.
5. Theseareafewofits forms:

StringBufferinsert(intindex,Stringstr)

StringBuffer insert(int index, char ch)

StringBufferinsert(intindex,Objectobj)

Here,indexspecifiestheindexatwhichpointthestringwillbeinsertedintothe invoking StringBuffer object.

6. Thefollowingsampleprograminserts“like”between“I”and“Java”:

```
classinsertDemo{publicstaticvoidmain(Stringargs[]){
 StringBuffer sb = new StringBuffer("I Java!");
 sb.insert(2, "like ");
```

```
        System.out.println(sb);
    }
  }
```

7. Output

   IlikeJava!

H. reverse()

YoucanreversethecharacterswithinaStringBufferobjectusingreverse(),shownhere: StringBuffer reverse( )

Thismethodreturnsthereversedobjectonwhichitwascalled. The following program demonstrates reverse( )

```
classReverseDemo{
publicstatic void main(Stringargs[])
{StringBuffers=newStringBuffer("abcdef");
System.out.println(s);
s.reverse();
System.out.println(s);
}
}
```

Output

abcdef

fedcba

I.delete()and deleteCharAt( )

YoucandeletecharacterswithinaStringBufferbyusingthemethodsdelete()and deleteCharAt( ).

Syntax:

StringBufferdelete(intstartIndex,intendIndex)

StringBuffer deleteCharAt(int loc)

Thedelete() method deletes asequenceof characters from theinvokingobject.

Here,startIndexspecifiestheindexofthefirstcharactertoremove,andendIndexspecifies an index one past the last character to remove.

Thus,thesubstringdeletedrunsfromstartIndextoendIndex–1.TheresultingStringBuffer object is returned.

ThedeleteCharAt()methoddeletesthecharacterattheindexspecifiedbyloc.Itreturnsthe resulting StringBuffer object.

//Demonstratedelete()anddeleteCharAt()

classdeleteDemo{ publicstatic voidmain(String args[])

{StringBuffersb=newStringBuffer("Thisisatest."); sb.delete(4,

7);

System.out.println("Afterdelete:"+sb);

sb.deleteCharAt(0);

System.out.println("AfterdeleteCharAt:"+sb);

}

}

Output

After delete: This a

test.AfterdeleteCharAt:hisate

st.

J. **replace()**

a. YoucanreplaceonesetofcharacterswithanothersetinsideaStringBufferobjectby calling replace( ).
b. Syntax

StringBufferreplace(intstartIndex,intendIndex,Stringstr)

Thesubstringbeing replaced isspecified bytheindexesstartIndexand endIndex.

c. Thus,thesubstringatstartIndexthroughendIndex–1isreplaced.Thereplacementstring is passed in str.

TheresultingStringBufferobjectisreturned.

class replaceDemo {

publicstatic void main(Stringargs[])

{StringBuffersb=newStringBuffer("Thisisatest."); sb.replace(5,

7, "was");

System.out.println("Afterreplace:"+sb);

```
}
```

}

Hereistheoutput:

Afterreplace:Thiswasa test.

K.**substring()**

1. Ithasthefollowingtwoforms:
   Syntax
   Stringsubstring(int startIndex)

   Stringsubstring(intstartIndex,int endIndex)

2. Thefirstformreturnsthesubstringthatstartsatstart Indexandrunstotheendofthe invoking StringBuffer object.

3. Thesecondformreturnsthesubstringthatstartsatstart Indexandrunsthroughend Index–1. These methods work just like those defined for String that were described earlier.

DifferencebetweenStringBufferandStringBuilder.

1. J2SE5addsanewstringclasstoJava'salreadypowerfulstringhandlingcapabilities. This new class is called StringBuilder.
2. ItisidenticaltoStringBufferexceptforoneimportantdifference:itisnot synchronized, which means that it is not thread-safe.
3. TheadvantageofStringBuilderisfasterperformance.However,incasesinwhichyou are using multithreading, you must use StringBuffer rather than StringBuilder.

AdditionalMethodsinStringwhichwasincludedinJava5

1. **intcodePointAt(inti)**

Returnsthe Unicodecodepoint at thelocation specified byi.

2. **intcodePointBefore(int i)**
   Returnsthe Unicodecodepoint atthelocationthat precedes thatspecifiedby i.
3. **intcodePointCount(intstart,intend)**
   Returnsthenumberofcodepointsintheportionoftheinvoking Stringthatare between start and end– 1.
4. **booleancontains(CharSequencestr)**
   Returnstrueiftheinvokingobject containsthestringspecifiedbystr.Returnsfalse, otherwise.
5. **booleancontentEquals(CharSequencestr)**
   Returnstrueiftheinvokingstringcontainsthesamestringasstr.Otherwise,returns false.
6. **booleancontentEquals(StringBufferstr)**
   Returnstrueiftheinvokingstringcontainsthesamestringasstr.Otherwise,returns false.
7. **staticStringformat(Stringfmtstr,Object...args)**

Returnsastring formattedas specified byfmtstr.

8. **staticStringformat(Localeloc,Stringfmtstr,Object...args)**
   Returnsastring formattedas specified byfmtstr.
9. **booleanmatches(string regExp)**
   ReturnstrueiftheinvokingstringmatchestheregularexpressionpassedinregExp. Otherwise, returns false.
10. **intoffsetByCodePoints(intstart,intnum)**
    Returnstheindex withtheinvokingstringthatisnumcodepointsbeyondthestarting index specified by start.
11. **StringreplaceFirst(StringregExp,StringnewStr)**
    Returnsastringinwhichthefirstsubstringthatmatchestheregularexpression specified by regExp is replaced by newStr.
12. **StringreplaceAll(StringregExp,StringnewStr)**
    Returnsastringinwhichallsubstringsthatmatchtheregularexpressionspecifiedby regExp are replaced by newStr
13. **String[]split(StringregExp)**
    Decomposestheinvokingstringintopartsandreturnsanarraythatcontainsthe result. Each part is delimited by the regular expression passed in regExp.
14. **String[]split(StringregExp,intmax)**
    Decomposes the invoking string into parts and returns an array that contains the result. Each part is delimited bythe regular expression passed in regExp. The number of pieces is specified by max. If max is negative, then the invoking string is fully decomposed.Otherwise,ifmaxcontainsanonzerovalue,thelastentryinthereturned arraycontains the remainder ofthe invokingstring. If max is zero, theinvokingstring is fully decomposed.
15. **CharSequencesubSequence(intstartIndex,intstopIndex)**
    Returnsasubstringoftheinvokingstring,beginningatstartIndexandstoppingat stopIndex . This method is required bythe CharSequence interface, which is now implemented by String.

AdditionalMethodsinStringBufferwhichwasincludedinJava5

StringBufferappendCodePoint(intch)

AppendsaUnicodecodepointtotheendoftheinvokingobject.Areferencetothe object is returned.

intcodePointAt(inti)

Returnsthe Unicodecodepoint at thelocation specified by i.

intcodePointBefore(inti)

Returnsthe Unicodecodepoint atthelocationthat precedes thatspecifiedby i.

intcodePointCount(intstart,intend)

ReturnsthenumberofcodepointsintheportionoftheinvokingStringthatare between start and end– 1.

intindexOf(Stringstr)

SearchestheinvokingStringBufferforthefirstoccurrenceofstr.Returnstheindexof the match, or –1 if no match is found.

intindexOf(String str, intstartIndex)

SearchestheinvokingStringBufferforthefirstoccurrenceofstr,beginningat startIndex. Returns the index of the match, or –1 if no match is found.

intlastIndexOf(Stringstr)
SearchestheinvokingStringBufferforthelastoccurrenceofstr.Returnstheindexof the match, or –1 if no match is found.

intlastIndexOf(Stringstr,intstartIndex)
SearchestheinvokingStringBufferforthelastoccurrenceofstr,beginningat startIndex. Returns the index of the match, or –1 if no match is found.

Servlet

Introdutiontoservlet

Servlet is small program that execute on the server side of a web connection. Just as appletextendthefunctionalityofwebbrowsertheappletextendthefunctionalityofwebserver.

Inordertounderstandtheadvantagesofservlet,youmusthavebasicunderstandingof how web browser communicates with the web server.

Considerarequestforstaticpage.AuserentersaURLintobrowser.Thebrowsergeneratesthttp request to a specific file. The file is returned by http response. Web server map this particular request for this purpose. The http header in the response indicates the content. Source of web page as MIME type of text/html.

1. **WhataretheAdvantageofServletOver"Traditional"CGI?**

Javaservletismore efficient, easierto use, morepowerful,moreportable,andcheaperthantraditionalCGIandthanmanyalternativeCGI-liketechnologies.(Moreimportantly,servletdevelopers get paid more than Perl programmers :-).

- **Efficient.** With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominatetheexecutiontime.Withservlets,theJavaVirtualMachinestaysup,andeach request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are *N* simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are *N* threads but only a single copy of the servlet class.
- **Convenient.** Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automaticallyparsinganddecodingHTMLformdata,readingandsettingHTTPheaders, handling cookies, tracking sessions, and many other such utilities.
- **Powerful.** Java servlets let you easily do several things that are difficult or impossible withregularCGI.Foronething,servletscantalkdirectlytotheWebserver(regularCGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain

informationfromrequesttorequest,simplifyingthingslikesessiontrackingandcaching of previous computations.

- **Portable.**ServletsarewritteninJavaandfollowawell-standardizedAPI.Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or Web Star. Servlets are supported directly or via a plug in on almost every major Web server.
- **Inexpensive.** There are a number of free or very inexpensive Web servers available that aregoodfor"personal"useorlow-volumeWebsites.However,withthemajorexception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servletsupportto it(if itdoesn'tcome preconfigured to supportservlets) is generally free or cheap.

2. Whatisservlet?Whatarethephasesofservletlifecycle?Givean example.

Servlets are small programs that execute on the server side of a web connection. Just as appletextendthefunctionalityofwebbrowsertheappletextendthefunctionalityofwebserver.

Servletclassis loaded.

Servletclassisloadedwhenfirstrequesttoweb container.

servletinstanceis created:

Webcontainercreatestheinstanceofservletclassonlyonce.

initmethodis invoked:

Itclasstheinitmethodwhenitloadstheinstance.Itisusedtointialiseservlet.

Syntaxofinitmethodis

publicvoidinit(ServletConfigconfig)throwsServletException

Servicemethodis invoked:

Webcontainercallsservicemethodeachtimewhenrequestfortheservletis received. Ifservlet is not initialized it calls init then it calls the service method.Syntax of service method is as follows

publicvoidservice(Servletrequest,ServletResponseresponse)throwsServletException, IOException

Destroymethodisinvoked.

Thewebcontainercallsthedestroymethodbeforeitremovestheservletfrom service. It gives servlet an opportunity to clean up memory, resources etc. Servlet destroy method has following syntax.

publicvoid destroy().

Ready

Therearethreestatesofservletnew,ready,end.Itisinnewstatewhenservletis created. Theservletinstanceiscreatedwhenitisinnewstate.Afterinvokingtheinit()methodservlet comes to ready state. In ready state servlet invokes destroy method it comes to end state.

3. Explainaboutdeploymentdescriptor

DeploymentdescriptorisafilelocatedintheWEB-INFdirectorythatcontrolsthe behavior of a java servlet and java server pages. The file is called the web.xml file and contains the header, DOCTYPE, and web app element. The web app element should contain a servlet element with three elements. These are servlet name, servlet class, and init-param.

Theservletnameelementscontainthenameusedtoaccessthejavaservlet.Theservlet class is the name of the java servlet class. init-param is the name of an initialization parameter that is used when request is made to the java servlet.

Examplefile:

<?xmlversion=”1.0”encoding-“ISO-8859=1”?>…..XMLheader

<!DOCTYPEweb-appPUBLIC“~//SunMicrosystems,Inc.??DTDWeb Application2.2//EN”> ..doctype

<web-app>

<servlet>

<servlet-name>MyJavaservlet</servlet-name>

<servlet-class>myPackage.MyJavaservletClass</servlet-class>

<init-param><param-name>parameter1</param-name>

<param-value>735</param-value>

</init-param>

</servlet>

</web-app>

4. **Howtoread datafromclientin servlet?**

- A client uses either the GET or POST method to pass information toajavaservlet.Dependingonthemethodusedbytheclienteither doGet() or doPost() method is called in servlet.

- Data sent by a client is read into java servlet by calling getParameter() method of HttpservletRequest() object that instantiatedintheargumentlistofdoGet()methodanddoPost() method.

- getParameter() requires one argument, which is the name of parameterthatcontainsthedatasentbytheclient.getParameter() returns the String object.

- String object contains the value assigned by the client. An empty string object is returned when it does not assign a value to the parameter.Alsoanullisretunedwhenparameterisnotreturnedin the client.

- getParameterValues()usedtoreturnthearrayofstring objects.

Example code

Htmlcodethatcallsa servlet:

<FORMACTION=”/servlet/myservlets.js2”>
EnterEmailAddress:<INPUTTYPE=”TEXT”NAME=”email”>
<INPUTTYPE=”SUBMIT”>
</FORM>

```
import java.io.*;
importjavax.servlet.*;
import javax.servlet.http.*;
publicclassjs2extendsHttpservlet{
publicvoiddoGet(HttpservletRequsetrequest,HttpservletResponseresponse)
throws servletException , IOException {
//Stringemail;
//Email=request.getParameter("email");
Respose.setContentType("text/html");
PrinterWriterpw=response.getWriter();
pw.println("<HTML>\n" +
"HEAD><TITLE>JavaServlet</TITLE></HEAD>\n"+
"<BODY>\n"+
//"<p>MYEmailAddress:"+email+"</p>\n"+
<h1>MyFirstServlet
"</BODY>\n" +
</HTML>");
}
}
```

5. **HowtoreadHTTPRequest Headers?**

A request from client contains two components these are implicit data, suchasemailaddressexplicitdataatHTTPrequestheader.Servletcanreadthese request headers to process the data component of the request.

ExampleofHTTPheader:

Accept:image.jpg,image.gif,*/*
Accept- Encoding: Zip
Cookie: CustNum-12345
Host:www.mywebsite.com
Referer:http://www.mywebsite.com/index.html

TheusesofHTTP header:

Accept:Identifiesthemailextension
Accept-Charset:Identifiesthecharactersetthatcanbeusedbybrowser. Cookie returns the cookies to server.
Host:containshostportal.
Referrer:ContainstheURLofthewebpagethatiscurrentlydisplayedin the browser.

AjavaservletcanreadanHTTPrequestheaderbycallingthegetHeader() method of the HttpservletRequest object. getHeader() requires one argument which is the name of the http request header.

getHeader()

6. **HowtosenddatatoclientandwritingtheHTTPResponse Header?**

AjavaservletrespondstoaclientsrequestbyreadingclientdataandHTTP request headers, and then processing information based on the natureof the request.

For example, a client request for information about merchandise in an online productcatalogrequiresthejavaservlettosearchtheproductdatabasetoretrieveproduct information and then format the product information into a web page, which is returned to client.

Therearetwowaysinwhichjavaservletrepliestoclientrequest.Thesearesent bysendinginformationtotheresponsestreamandsendinginformationinhttpresponse header. The http response header is similar to the http request header.

ExplicitdataaresentbycreatinganinstanceofthePrintWriterobjectandthen using println() method to transmit the information to the client.

Implcitdataexample:HTTP/1.1200OK
Content-Type:text/plain

My Response

JavaservletcanwritetotheHTTPresponseheaderbycallingsetStatus()method requires one argument which is an integer that represent the status code.

```
Response.setStatus(100);
```

7. **ExplainaboutCookiesin servlet.**

Cookiesaretextfilesstoredontheclientcomputerandtheyarekeptforvarious information tracking purpose. Java Servlets transparently supports HTTP cookies.

Therearethreestepsinvolvedinidentifyingreturningusers:

- Serverscriptsendsasetofcookiestothebrowser.Forexamplename,age,or identification number etc.
- Browserstoresthisinformationonlocalmachineforfutureuse.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

SettingCookieswithServlet:

Settingcookieswithservletinvolvesthreesteps:

(1) Creating aCookieobject: YoucalltheCookieconstructor withacookienameand acookie value, both of which are strings.

```
Cookiecookie=newCookie("key","value");
```

(2) Settingthemaximumage: YouusesetMaxAgetospecifyhowlong(inseconds)thecookie should be valid. Following would set up a cookie for 24 hours.

cookie.setMaxAge(60*60*24);

(3) Sending the Cookie into the HTTP response headers: You use **response.addCookie** toadd cookies in the HTTP response header as follows:

response.addCookie(cookie);

Writing

Cookieimport

java.io.*;

importjavax.servlet.*;

import javax.servlet.http.*;

public class HelloForm extends HttpServlet {

publicvoiddoGet(HttpServletRequestrequest,

HttpServletResponse response)

throwsServletException,IOException

{

CookiemyCookie=newCookie("userid",123);

myCookie.setMaxAge(60*60);

response.addCookie(myCookie);

response.setContentType("text/html");PrintWriter

out = response.getWriter();

out.println("<html>\n"+

"<head><title>"+MyCookie+"</title></head>\n"+

"<nody>\n" +

"<h1>+ My Cookie +"<h1>\n" +

" CookieWritten+ \n"+

"</body></HTML>");

}

```
}
```

ReadingCookieswith Servlet:

To read cookies, you need to create an array of *javax.servlet.http.Cookie* objects by calling the **getCookies( )** method of *HttpServletRequest*. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

Example:Letusreadcookieswhichwehavesetinpreviousexample:

```
importjava.io.*;importjavax.servlet.*;importjavax.servlet.http.*;
 public class ReadCookies extends HttpServlet {
 publicvoiddoGet(HttpServletRequestrequest,HttpServletResponseresponse) throws
        ServletException, IOException
 {
    Cookie cookie;
        Cookie[]cookies;
    cookies = request.getCookies();
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Stringtitle="ReadingCookiesExample";
    out.println( "<html>\n" +
          "<head><title>"+title+"</title></head>\n"); if(
               cookies != null ){
         out.println("<h2>FoundCookiesNameandValue</h2>"); for
          (int i = 0; i < cookies.length; i++){
            cookie=cookies[i];
           out.print("Name:"+cookie.getName()+",");
       out.print("Value: " + cookie.getValue( )+"<br/>");
      }
```

```
}else{    out.println("<h2>Nocookiesfounds</h2>");
}    out.println("</body>");    out.println("</html>");
}}
```

8. ExplainSessionTracking:

1. A session is created each time a client requests service from a java servlet. The java. The java servlet processes the request and response accordingly, after which thesessionisterminated.Manytimesthesameclientfollowswithanotherrequest to the same client follows with another request to the same java servlet, java servlet requires information regarding the previous session to process request.
2. However,HTTPisstatelessprotocol,meaningthatthereisnotholdoverfrom the previous sessions.
3. Java servlet is capable of tracking sessions by using Httpsession API.It determinesiftherequestisacontinuationfromanexistingsessionornew session.
4. Ajava servlet callsa getSession()method ofHttpservletRequset object, which returnsasessionobjectifitisanewsession.ThegetSession()methodrequires one argument which is Boolean true. Returns session object.

Syntax:

HttpSessions1=request.getSession(true);

JSP program

Ajspisjavaserverpageisserversideprogramthatissimilarindesignandfunctionality to a java servlet.

AJSPiscalledbyaclienttoprovidewebservices,thenatureofwhichdependsonclient application.

AjspissimplertocreatethanajavaservletbecauseajspiswritteninHTMLratherthan with the java programming language. . There are three methods that are automatically called when jsp is requested and when jsp terminates normally. These are the jspInit() method , the jspDestroy()m ethod and service() method.

AjspInit()isidenticaltiinit()methodofjavaservlet.Itiscalledwhenfirsttimejspis called.

A jspDestroy()is identical to destroy() method of servlet. The destroy() method is automaticallycalledwhenjsperminatesnormally.Itisnotcalledwhenjspterminatesabruptly.It is used for placing clean up codes.

1. **ExplainJSPtags(repeatedquestion)**

AjsptagconsistsofacombinationofHTMLtagsandJSPtags.JSPtagsdefine java code that is to be executed before the output of jsp program is sent to the browser.

Ajsptagbeginwitha<%,whichisfollowedbyjavacode,andwndswith%>, There ia an

XML version of jsp tag <jap:TagId></jsp:TagId>

AjsptagsareembeddedintotheHTMLcomponentofajspprogramandare processed by Jsp virtual engine such as Tomcat.

Javacodeassociatedwithjsptagareexecutedandsenttobrowser. There

are five types of jsp tags :

Commenttag:Acommenttagopenswith<%--andclosewith--%>andisfollwedbya comment that usually describes the functionality of statements that follow a comment tag.

Declaration statement tags:A declartion statement tag opens with <%!And is followedbydeclarationstatementsthatdefinethevariables,object,andmethodsthatareavilabe to other component of jsp program.

Directivetags: Adirectivetagopenswith<%@andcommandsthejspvirtualengineto perform a specific task, such as importing java package required by objects and methods used in a declaration statement. The directive tag closes with %> . There are commonly used in directivesimport,include,andtaglib.Theimporttagisusedtoimportjavapackagesintothejsp program. Include is used for importing file. Taglib is used for including file.

Example:

<%@pageimport=”importjava.sql.*”; %>

<%@includefile=”keogh\books.html”%>

<%@tagliburl=”myTags.tld”;%>

Expression tags: An expression tag opens with <%= and is used for an expression statementwhoseresultpagereplacestheexpressiontagwhenthejspvirtualengineresolvesJSP tags. An expression tag closes with %>

Scriptlettag: Asciptlettagopenswith<%andcontainscommonlyusedjavacontrol statements and loops. And Scriptlet tag closes with %>

2. **HowvariablesandobjctsdeclaredinJSPprogram?**

   Youcandeclarejavavariablesandobjectsthatareusedina JSPprogramby using the same codin technique used to declare them in java.JSP declaration statements must appear as jsp tag

   Ex:

   <html>

   <head>

   <title>JspProgramming</title>

   <.head>

   <body>

   <%!Intage=29;%> Yourageis:<%=age%>

   </body>

   </html>

3. **Howmethodaredeclaredandusedinjsp programs?**

   Methodsaredefinedsamewayasitisdefinedinjspprogram,exceptthese are placed in JSP tag.methods are declared in JSP decalration tag. The jspcalls method in side the expression tag.

   Example:

```
<html>
<head>
    <title>Jsp programming</title>
</head>
<body>
<%!intadd(intn1,intn2)
    {
    int c;
    c=a+b;
    returnc;
```

}

%>

** Additionoftwonumbers:<%=add(45,46)%> **

</body></html>

4. **ExplainthecontrolstatementsofJSPwithexampleprogram:**
   1. OneofthemostpowerfulfeaturesavilableinJSPistheabilitytochangethe flow of the program to truly create dynamic content for a web based on conditions received from the browser.
   2. Theraretwocontrolstatementsusedtochangetheflowofprogramare“if“ and “switch” statement , both of which are also used to direct the flow of a java program.

      Ex:

```
<html>
<head>
        <title>JSPProgramming</title>
</head>
<body>
        <%!intgrade=26;%>
        </body>
<%if(grade>69){%>
        <p>YouPassed!</p>
<%} else{ %>
<p>BetterLuckNextTime</p>
<% } %>
</body>
</html>
```

5. **LoopingStatementofJSP**

   Jsploopsarenearlyidenticaltoloopsthatyouuseinyourjavaprogram except you can repeat the html tags

   Therearethreekindofjsploopthatarecommonlyusedinjspprogram. Ex:for loop , while loop , do while .

   LoopplaysanimportantroleinJSPdatabaseprogram.Thefollowing program is example for “FOR LOOP”.

<html><head><title>ForLoopExample</title></head>

<body>

```
<%
   for(inti=0;i<10;i++){
%>
<p>HelloWorld</p>
<%}%></body>
</html>
```

6. **ExplainRequsetStringgeneratedbybrowser.howtoreadarequeststringin jsp?**
   1. Abrowsergeneraterequststringwheneverthesubmitbuttonisselected.The user requests the string consists of URL and the query the string.
      Example of request string:
      http://www.jimkeogh.com/jsp/?fname=”Bob”&lname=”Smith”
   2. Yourjspprogramneedstoparsethequerystringtoextractthevaluesoffields that are to be processed by your program. You can parse the query string by using the methods of the request object.
   3. getParameter(Name)methodusedtoparseavalueofaspecificfieldthatare to be processed by your program
   4. codetoprocesstherequsetstring
      <%! String FirstName =requst.getParameter(fname);
      StringLastName=requst.getParameter(lname);%>
   5. Copyingfrommultivaluedfieldsuchasselectionlistfieldcanbetricky multivalued fields are handled by using getParameterValues()
   6. Otherthanrequsetstringurlhasprotocols,portno,thehostname
   7. **WritetheJSPprogramtocreateandreadcookiecalled“EMPID”and that has value “AN2536”.**

Cookieissmallpieceofinformationcreatedba JSPprogramthatisstored on the clien’s hard dik by the browser. Cookie isare used to store various kind of information, such as user preference. The cookies are created by using Cookie class.

Createcookie:

```
<html>
<head>
<title>creating cookie</title>
</head>
<body>
<%!StringMyCookieName=”EMPID”;
```

```
        String UserValue=”AN2536”;
%>
</body>
</html>
```

ReadingCookie:

```
<html>
<head>
<title>readingcookie</title>
</head>
<body>
<%StringmyCookieName=”EMPID”;
String myCookieValue;
StringCName,CValue; int
found=0;
Cookie[] cookies=request.getCoookies();for(
        int i=0;i<cookies.length;i++) {
        CName= cookies[i].getName();
        CValue =cookies[i].getValue();
        If(myCookieName.equals(CName)){
        found=1;
        myCookieValue=Cvalue;}}
        If(found== 1){ %>
        <p>CookieName =<%=CName%></p>
        <p>CookieValue=<%=CValue%></p>
        <%}%></body></html>
```

8. **Explainstepstoconfiguretomcat.**
   i. JspprogramprogramsareexecutedbyaJSPvirtualmachinethat run on a webserver.
   ii. WecandownloadandinstallJSPvirtualmachine.
   iii. InstallationSteps
        ConnecttoJakarta.apache.org.
        Select down load
        SelectBinariestodisplaythebinaryDownloadPage.
        Createa folderfrom therootdirectory calledtomcat.
        Download latest release.
        UnzipJakarta-tomcat.zip.
        Theextractionprocesscreatesthefollowingfolderinthe tomcat directory: bin, conf, doc, lib, src, and webapps
        Modify the batch file , which is located in the \tomcat\bin folder.ChangetheJAVA_HOMEvariableisassignedthepathewhere JDK is installed on your computer.
        Opendoswindowandtype\tomcat\bin\tomcattostart Tomcat.
        Openyourbrowser.Enterhttp://localhost:8080.

Tomcathomepageisdisplayedonthescreenverifyingthat Tomcatis running.

9. Explainhowsessionobjectsare created.

AJSPdatabasesystemiaabletoshareinformationamongJSPprograms within a session by using a session object. Each time a session is created , a unique IDis assigned to the session and stored as acookie.

AuniqueIDenablesJSPprogramtotrackmultiplesessionsession simultaneouslywhilemaintainingdataintegrityofeachsession.ThesessionIDisusedto prevent the intermingling of each session.

CreatesessionObject:

```
<html><head><title>JspSession</title></head>

<body>

<%!StringAtName=”Product”;

    String AtValue =”1234”;

    Session.setAttributes(AtName,AtValue);

%></body></html>
```

Insessionobjectwecanstoreinformationaboutpurchasesassessionattributes can be retrived and modified each time the jsp program runs. setAttributes() used for creating attributes.

ReadSessionObject:

getAttributeNames()methosreturnsnamesofalltheattribbutesasEnumeration, the attributes are processsed.

```
<html><head><title>JspSession</title></head>

<body><%!

    Enumaration purchases=session.getAttributeNames();

    String AtName=(String) attributeNames.nextElement();

    StringAtValu=(String)session.getAttribute(AtName);%>

    <p>AttirbuteName<%=AtName%></p>

    <p>AttributeValue<%=Atvalue%></p>

    <%}%>%></body></html>
```

IIIInternalQuestions

1. WhataredifferenttypesofJSPtagsdescribetheJSPtagswithexample.(Dec2011)
2. DefineJSP.Explaintwotypesofcontrolstatementswithexample.(Dec 2012)
3. WritetheJSPprogramtocreateandreadcookiecalled“EMPID”andthathasvalue “AN2536”(Dec 2012)
4. WhatisRMI?BrieflyexplainworkingofRMIinjava.(Dec2012)
5. Departmenthassetthegradeforthesubject Javaas follows:

   Above90:A, 80- 89:B,70-79 :C
   Below70=fail.Shamentershismarksforthesubject Javaintheinterfaceprovided. Write a JSP program to accept the mark and display the grade.(Jun 2011)

6. BrieflyexplaintheRMIinJava(June2011)

7. DiscussdifferenttypesofJSPtags(Jun2011)

8. WriteaprogramusingRMIsuchasclientandserverprograminwhichclientsendshello message to server and replies to client (June 2012)

9. DevelopsimplejavaservletthathandleHTTPRequestandResponse(June 2012)

10. Explainjavax.servletpackages(June2012)

11. WhatisdifferencebetweenJSPandServlet?(june2012)

12. WhataretheadvantagesofJSPprogram?(jun 2010)

13. Whatareservlets?Brieflyexplaintheapplicationofservletsinwebprogramming(dec 2010)

14. Explainthelifecycleofaservlet.(dec2010)
15. Writeajavaservletwhichreadstwoparametersfromthewebpage,sayvalue1andvalue 2,whicharetypeintegerandfindsthesumofthetwovalueandreturnbacktheresultas a webpage.(dec 2010)

16. Providejavasyntaxforthefollowing:(dec 2010)
    i) HandlingHTTPrequestsand responses
    ii) Using cookies
    iii) Session tracking
17. ListoutdifferencebetweenCGIandservlet.
18. Whatiscookielistoutmethodsdefinedbycookie.Writeaservletprogramtoread cookie.
19. Writeajspprogramtoaddcookiename“UserId”and value”JB007”
20. Describeindetailhowtomcatwebserverisconfiguredindevelopofservletlife cycle.