

Containers - Part I

Suggested Reading:

Bruce Eckel, Thinking in Java (Fourth Edition)
Holding Your Objects (Collection, Iterator, ArrayList)

Interface Collection

Method	Description (from docs API)
boolean add(Object o)	ensures that this collection contains the specified element
boolean addAll(Collection c)	adds all of the elements in the specified collection to this collection
void clear()	removes all of the elements from this collection
boolean contains(Object o)	returns true if this collection contains the specified element
boolean containsAll(Collection o)	returns true if this collection contains all of the elements in the specified collection
boolean equals(Object o)	compares the specified object with this collection for equality

Dr. Cristina Marinescu

2

Interface Collection

Method	Description (from docs API)
int hashCode()	returns the hash code value for this collection
boolean isEmpty()	returns true if this collection contains no elements
Iterator iterator()	returns an iterator over the elements in this collection
boolean remove(Object o)	removes a single instance of the specified element from this collection, if it is present
boolean removeAll(Collection c)	removes all of this collection's elements that are also contained in the specified collection
boolean retainAll(Collection c)	retains only the elements in this collection that are contained in the specified collection

Dr. Cristina Marinescu

3

Interface Collection

Method	Description (from docs API)
int size()	returns the number of elements in this collection
Object[] toArray()	returns an array containing all of the elements in this collection
Object[] toArray(Object[] a)	returns an array containing all of the elements in this collection

import java.util.*;

Dr. Cristina Marinescu

4

Class ArrayList

Constructor Description (from docs API)

ArrayList() constructs an empty list with an initial capacity of ten

ArrayList(Collection c) constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator

ArrayList(int initialCapacity) constructs an empty list with the specified initial capacity

Dr. Cristina Marinescu

5

```
import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList = new ArrayList(3);

        stringList.add("String1");
        stringList.add("String2");
        stringList.add(new String("String1"));
        stringList.add("String3");

        System.out.println("Collection size " + stringList.size());
        System.out.println("Collection content " + stringList);
    }
}

Collection size 4
Collection content [String1, String2, String1, String3]
```

Dr. Cristina Marinescu

6

```
import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList = new ArrayList(3);

        Collection size 4
        Collection content [String1, String2, String1, String3]

        stringList.addAll(stringList);
        System.out.println("Collection size " + stringList.size());
        System.out.println("Collection content " + stringList);
        System.out.println("Collection contains String2 "
                           + stringList.contains("String2"));

    }
}

Collection size 8
Collection content [String1, String2, String1, String3, String1, String2, String1, String3]
Collection contains String2 true
```

Dr. Cristina Marinescu

7

```
import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList = new ArrayList(3);

        Collection size 8
        Collection content [String1, String2, String1, String3, String1, String2, String1, String3]
        Collection contains String2 true

        ArrayList stringList2 = new ArrayList(3);
        stringList2.addAll(stringList);
        System.out.println("Collection2 content " + stringList2);
        System.out.println("Collection2 isEmpty " + stringList2.isEmpty());
    }
}

Collection2 content []
Collection2 isEmpty true
```

Dr. Cristina Marinescu

8

```

import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList = new ArrayList(3);

Collection size 8
Collection content [String1, String2, String1, String3, String1, String2, String1, String3]
Collection contains String2 true
Collection2 content []
Collection2 isEmpty true

        stringList2.addAll(stringList);
        System.out.println("Collection2 content " + stringList2);
        System.out.println("Collection2 isEmpty " + stringList2.isEmpty());
        stringList2.remove("String2");
        System.out.println("Collection2 content " + stringList2);
        stringList2.removeAll(stringList);
        System.out.println("Collection2 isEmpty " + stringList2.isEmpty());
    }
}
Collection2 content [String1, String2, String1, String3, String1, String2, String1, String3]
Collection2 isEmpty false
Collection2 content [String1, String1, String3, String1, String2, String1, String3]
Collection2 isEmpty true

```

Dr. Cristina Marinescu 9

```

import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList2 = new ArrayList(3);

        stringList2.add("String4");
        System.out.println("Collection2 content " + stringList2);
        stringList2.clear();
        System.out.println("Collection2 isEmpty " + stringList2.isEmpty());
    }
}

```

Collection2 content [String4]
Collection2 isEmpty true

Dr. Cristina Marinescu 10

```

import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList2 = new ArrayList(3);

        stringList2.add("String1");
        stringList2.add("String2");
        stringList2.add("String3");
        stringList2.add("String4");

        Object[] o1 = stringList2.toArray();
        Object[] o2 = new Object[4];
        System.out.println("o2 " + o2);
        o2 = stringList2.toArray(o2);
        System.out.println("o2 " + o2);
    }
}
o2 [Ljava.lang.Object;@41675ec4
o2 [Ljava.lang.Object;@41675ec4

```

Dr. Cristina Marinescu 11

```

import java.util.ArrayList;

class Test {
    public static void main(String[] args) {
        ArrayList stringList2 = new ArrayList(3);

        stringList2.add("String1");
        stringList2.add("String2");
        stringList2.add("String3");
        stringList2.add("String4");

        Object[] o2 = new Object[3];
        System.out.println("o2 " + o2);
        o2 = stringList2.toArray(o2);
        System.out.println("o2 " + o2);
    }
}
o2 [Ljava.lang.Object;@41675ec4
o2 [Ljava.lang.Object;@697eb767

```

Dr. Cristina Marinescu 12

```

import java.util.ArrayList;

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        System.out.println("Size " + bookList.size());
        System.out.println("Collection Contains " + bookList.contains(b1));
        System.out.println("Collection Contains " + bookList.contains(b2));
        System.out.println("Collection Contains " + bookList.contains(new Book(1)));
    }
}

```

Dr. Cristina Marinescu 13

```

import java.util.ArrayList;

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public boolean equals(Book b) {
        return b.pages == pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        System.out.println("Size " + bookList.size());
        System.out.println("Collection Contains " + bookList.contains(b1));
        System.out.println("Collection Contains " + bookList.contains(b2));
        System.out.println("Collection Contains " + bookList.contains(new Book(1)));
    }
}

```

Dr. Cristina Marinescu 14

```

import java.util.ArrayList;

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public boolean equals(Object b) {
        return (b instanceof Book)
            && (((Book)b).pages == pages);
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        System.out.println("Size " + bookList.size());
        System.out.println("Collection Contains " + bookList.contains(b1));
        System.out.println("Collection Contains " + bookList.contains(b2));
        System.out.println("Collection Contains " + bookList.contains(new Book(1)));
    }
}

```

15

```

import java.util.ArrayList;

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public boolean equals(Object b) {
        return (b instanceof Book)
            && (((Book)b).pages == pages);
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        System.out.println("Size " + bookList.size());
        System.out.println(bookList);
    }
}

```

Dr. Cristina Marinescu 16

```

import java.util.ArrayList;

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public String toString() {
        return "Book " + pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        System.out.println("Size " + bookList.size());
        System.out.println(bookList);
    }
}

```

Dr. Cristina Marinescu 17

**Do not forget to override
equals and toString!!!**

Dr. Cristina Marinescu 18

Towards Collection - equals(Object o)

```

class Test {
    public static void main(String[] args) {
        Book b11 = new Book(1);
        Book b12 = new Book(2);
        Book b21 = new Book(1);
        Book b22 = new Book(2);

        ArrayList bookList1 = new ArrayList();
        ArrayList bookList2 = new ArrayList();

        bookList1.add(b11); bookList1.add(b12);

        bookList2.add(b21); bookList2.add(b22);

        boolean areEquals = true;
        if(bookList1.size()!=bookList2.size())
            areEquals = false;
        else
            for(int i=0; i<bookList1.size(); i++)
                if(bookList1.get(i).equals(bookList2.get(i)) == false){
                    areEquals = false;
                    break;
                }
        System.out.println("Collections are equals " + areEquals);
    }
}

```

19

Towards Collection - equals(Object o)

```

class Test {
    public static void main(String[] args) {
        Book b11 = new Book(1);
        Book b12 = new Book(2);
        Book b21 = new Book(1);
        Book b22 = new Book(2);

        ArrayList bookList1 = new ArrayList();
        ArrayList bookList2 = new ArrayList();

        bookList1.add(b11); bookList1.add(b12);

        bookList2.add(b21); bookList2.add(b22);

        System.out.println("Collections are equals "
            + bookList1.equals(bookList2));
    }
}

```

Dr. Cristina Marinescu 20

```

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString() {
        return "Book " + pages;
    }
}

```

[Book 10, Book 10]

Fetching containers (a)

```

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

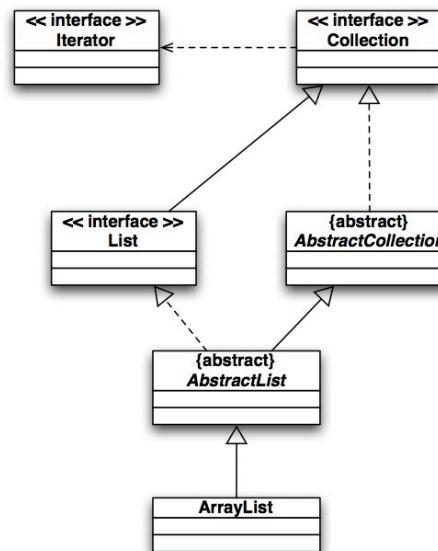
        for(Object o:bookList)
            ((Book)o).setPages(10);

        System.out.println(bookList);
    }
}

```

Dr. Cristina Marinescu

21



Dr. Cristina Marinescu

22

Interface List

Method	Description (from docs API)
Object get(int index)	returns the element at the specified position in this list
Object set(int index, Object element)	replaces the element at the specified position in this list with the specified element
int lastIndexOf(Object o)	returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element
boolean remove(Object o)	removes the first occurrence of the specified element from this list, if it is present

Dr. Cristina Marinescu

23

```

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString() {
        return "Book " + pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        for(int i=0; i<bookList.size(); i++)
            ((Book)bookList.get(i)).setPages(10);

        System.out.println(bookList);
    }
}

```

[Book 10, Book 10]

Dr. Cristina Marinescu

24

Fetching containers (c)

Interface Collection

Method	Description (from docs API)
--------	------------------------------

....

Iterator iterator()

returns an iterator over the elements in this collection

....

Dr. Cristina Marinescu 25

Method	Description (from docs API)
--------	------------------------------

boolean hasNext() returns true if the iteration has more elements

Object next() returns the next element in the iteration

void remove() removes from the underlying collection the last element returned by this iterator

Dr. Cristina Marinescu 26

```
class Book{  
    private int pages;  
  
    public Book(int p) {  
        pages = p;  
    }  
  
    public void setPages(int p){  
        pages = p;  
    }  
  
    public String toString() {  
        return "Book " + pages;  
    }  
}  
  
[Book 10, Book 10]
```

import java.util.Iterator;

Fetching containers (c)

```
Iterator it = bookList.iterator();  
while(it.hasNext())  
    ((Book)it.next()).setPages(10);
```

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

}

Dr. Cristina Marinescu 27

```
class Book{  
    private int pages;  
  
    public Book(int p) {  
        pages = p;  
    }  
  
    public void setPages(int p){  
        pages = p;  
    }  
  
    public String toString() {  
        return "Book " + pages;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Book b1 = new Book(1);  
        Book b2 = new Book(2);  
        ArrayList bookList = new ArrayList(3);  
  
        bookList.add(b1);  
        bookList.add(b2);  
  
        Iterator it = bookList.iterator();  
        while(it.hasNext())  
            ((Book)it.next()).setPages(10);  
  
        System.out.println(bookList);  
    }  
}
```

X **Iterator it;
it = new Iterator();**

```
Iterator it = bookList.iterator();  
while(it.hasNext())  
    ((Book)it.next()).setPages(10);
```

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

}

Dr. Cristina Marinescu 28

```

class Book{
    private int pages;

    public Book(int p) {           Exception in thread "main" java.lang.ClassCastException:
        pages = p;               X java.lang.String cannot be cast to Book
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString(){
        return "Book " + pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);
        bookList.add("OOP");

        Iterator it = bookList.iterator();
        while(it.hasNext())
            ((Book)it.next()).setPages(10);

        System.out.println(bookList);
    }
}

```

Dr. Cristina Marinescu 29

A problem...

Another (annoying) problem...

```

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString(){
        return "Book " + pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList bookList = new ArrayList(3);

        bookList.add(b1);
        bookList.add(b2);

        Iterator it = bookList.iterator();
        while(it.hasNext())
            ((Book)it.next()).setPages(10);

        System.out.println(bookList);
    }
}

```

Dr. Cristina Marinescu 30

Perform a cast

```

class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString(){
        return "Book " + pages;
    }
}

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList<Book> bookList = new ArrayList<Book>(3);

        bookList.add(b1);
        bookList.add(b2);
        bookList.add("OOP");
    }
}

```

Compilation Error!!!

Dr. Cristina Marinescu 31

Generics and type-safe containers

```

class Test {
    public static void main(String[] args) {
        Book b1 = new Book(1);
        Book b2 = new Book(2);
        ArrayList<Book> bookList = new ArrayList<Book>(3);

        bookList.add(b1);
        bookList.add(b2);

        for(Book b:bookList)
            b.setPages(20);

        for(int i=0; i<bookList.size(); i++)
            bookList.get(i).setPages(30);

        Iterator<Book> it = bookList.iterator();
        while(it.hasNext())
            it.next().setPages(10);
    }
}

```

Dr. Cristina Marinescu 32

Generics and type-safe containers

```
class Test {  
    public static void main(String[] args) {  
        Book[] books = new Book[3];  
        books[0] = new Book(1);  
        books[1] = new Book(2);  
  
        Object[] objects = books;  
        objects[2] = "OOP";  
  
        books[2].setPages(25); ←  
    }  
}  
  
Runtime Error!!!
```

Dr. Cristina Marinescu

33

Generics and type-safe containers

```
class Test {  
    public static void main(String[] args) {  
        ArrayList<Book> books = new ArrayList<Book>();  
        books.add(new Book(1));  
        books.add(new Book(2));  
  
        ArrayList<Object> objects;  
        objects = books; ←  
    }  
}
```

Compilation Error!!!

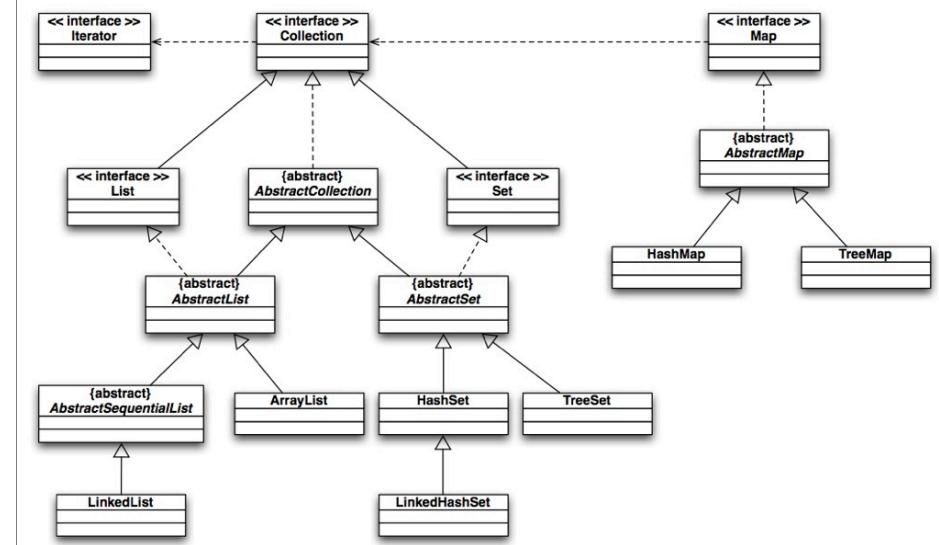
Dr. Cristina Marinescu

34

Containers - Part II

Suggested Reading:

Bruce Eckel, Thinking in Java (Fourth Edition)
Holding Your Objects (LinkedList, Set, Map)
Containers in Depth (Sets and storage order,
Understanding Maps, Hashing and hash codes)



Dr. Cristina Marinescu

2

```
import java.util.LinkedList;
import java.util.Iterator;
```

Class LinkedList

```
class Test {
    public static void main(String[] args) {
        LinkedList<Book> books = new LinkedList<Book>();
        books.add(new Book(1));
        books.add(new Book(2));
        System.out.println(books);

        for(Book b: books)
            b.setPages(15);
        System.out.println(books);

        Iterator<Book> it = books.iterator();
        while(it.hasNext())
            it.next().setPages(20);
        System.out.println(books);

        for(int i = 0; i < books.size(); i++)
            books.get(i).setPages(25);
        System.out.println(books);
    }
}
```

```
class Book{
    private int pages;

    public Book(int p) {
        pages = p;
    }

    public void setPages(int p){
        pages = p;
    }

    public String toString() {
        return "Book " + pages;
    }
}
```

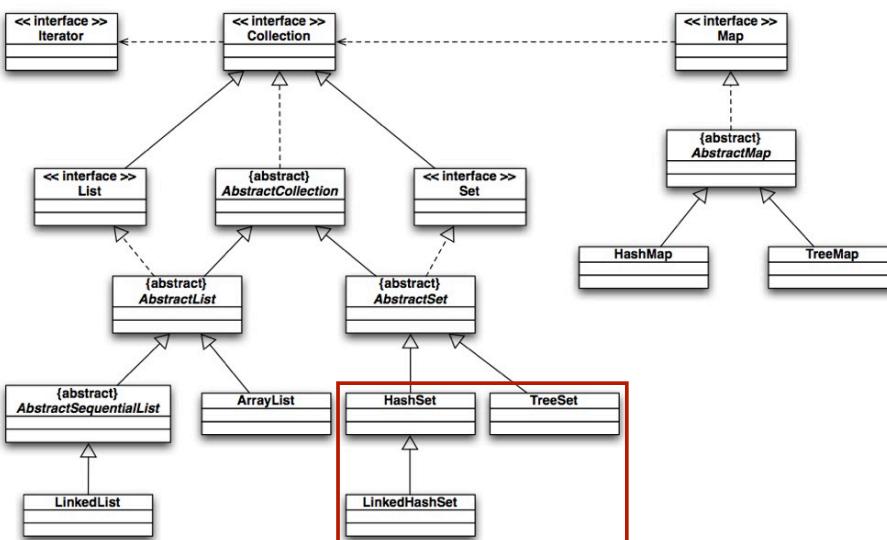
Dr. Cristina Marinescu 3

Class LinkedList

LinkedList inserts and removes elements in the middle of the list more efficiently than ArrayList.

LinkedList is less efficient for random-access operations than ArrayList.

Dr. Cristina Marinescu 4



Dr. Cristina Marinescu 5

```
import java.util.HashSet;

class JavaElementsSetMain {
    public static void main(String[] args){
        HashSet<String> set = new HashSet<String>();

        String s1 = "string1";
        String s2 = "string1";
        String s3 = new String("string1");

        set.add(s1);
        set.add(s2);
        set.add(s3);

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

prints

[string1]

Dr. Cristina Marinescu 6

```

public class MyElementsSet {
    public static void main(String[] args){
        MyElement m1 = new MyElement(3);
        MyElement m2 = new MyElement(4);
        MyElement m3 = new MyElement(1);
        MyElement m4 = new MyElement(1);

        HashSet<MyElement> set = new HashSet<MyElement>();
        set.add(m1);
        set.add(m2);
        set.add(m3);
        set.add(m4);
        System.out.println(set);
    }
}

prints ???

```

```

class MyElement {
    private int myValue;

    public MyElement(int v) {
        this.myValue = v;
    }

    public boolean equals(Object o){
        return ((o instanceof MyElement)
                && (myValue == ((MyElement)o).myValue));
    }

    public String toString() {
        return myValue + "";
    }
}

```

Dr. Cristina Marinescu 7

```

package sets;

class MyElement {
    private int myValue;

    public MyElement(int v) {
        this.myValue = v;
    }

    public boolean equals(Object o){
        return ((o instanceof MyElement)
                && (myValue == ((MyElement)o).myValue));
    }
}

public class MyElementsSet {
    public static void main(String[] args){
        MyElement m1 = new MyElement(3);
        MyElement m2 = new MyElement(4);
        MyElement m3 = new MyElement(1);
        MyElement m4 = new MyElement(1);

        System.out.println(m1 + " " + m1.hashCode());
        System.out.println(m2 + " " + m2.hashCode());
        System.out.println(m3 + " " + m3.hashCode());
        System.out.println(m4 + " " + m4.hashCode());
    }
}

prints

```

Dr. Cristina Marinescu 9

```

class JavaElementsSetMain {
    public static void main(String[] args){
        HashSet<String> set = new HashSet<String>();

        String s1 = "string1";
        String s2 = "string1";
        String s3 = new String("string1");

        set.add(s1);
        set.add(s2);
        set.add(s3);

        System.out.println(set);

        System.out.println(s1.hashCode());
        System.out.println(s2.hashCode());
        System.out.println(s3.hashCode());
    }
}

prints

```

-1881759168
-1881759168
-1881759168

Dr. Cristina Marinescu 8

public int hashCode() from class Object

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap.

The general contract of hashCode is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
- It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables. [from Java 1.7 API]

Dr. Cristina Marinescu 10

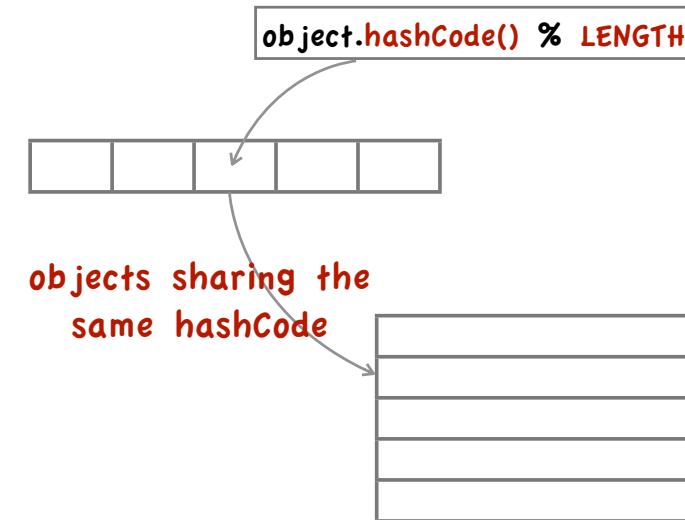
a recipe for generating hashCode is available @page 614

Bruce Eckel, Thinking in Java (Fourth Edition)
Containers in Depth

Dr. Cristina Marinescu 11

```
class MyElement {  
    private int myValue;  
  
    public MyElement(int v) {  
        this.myValue = v;  
    }  
  
    public boolean equals(Object o){  
        return (o instanceof MyElement)  
            && (myValue == ((MyElement)o).myValue);  
    }  
  
    public int hashCode() {  
        return myValue;  
    }  
  
    public String toString() {  
        return myValue + "";  
    }  
}  
  
Class HashSet  
[no guarantee regarding the iteration order]  
  
public class MyElementsSet {  
    public static void main(String[] args){  
        MyElement m1 = new MyElement(3);  
        MyElement m2 = new MyElement(4);  
        MyElement m3 = new MyElement(1);  
        MyElement m4 = new MyElement(1);  
  
        HashSet<MyElement> set = new HashSet<MyElement>();  
        set.add(m1);  
        set.add(m2);  
        set.add(m3);  
        set.add(m4);  
  
        System.out.println(set);  
    }  
}  
  
prints  
[1, 3, 4]
```

Dr. Cristina Marinescu 13



Dr. Cristina Marinescu 12

protected Object clone()
throws CloneNotSupportedException
from class Object

creates an object with a similar state as the object for which the clone() method was called.

Dr. Cristina Marinescu 14

protected Object clone() throws CloneNotSupportedException from class Object

```
class MyClass implements Cloneable {  
    private int myValue;  
    private ArrayList<Integer> myValues = new ArrayList<Integer>();  
  
    public void setValue(int myValue) { this.myValue = myValue; }  
  
    public void addInteger(Integer... ints) {  
        for(Integer i : ints) myValues.add(i);  
    }  
  
    public String toString() {  
        return myValue + " " + myValues.toString();  
    }  
  
    public ArrayList<Integer> getValues() { return myValues; }  
  
    public int getValue() { return myValue; }  
  
    public Object clone() throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

Shallow
Cloning

Dr. Cristina Marinescu 15

Shallow Cloning

```
public class CloningSample {  
    public static void main(String[] args) throws CloneNotSupportedException {  
        MyClass cls1 = new MyClass();  
        cls1.setValue(12);  
        cls1.addInteger(15, 17, 10);  
        System.out.println(cls1);  
  
        MyClass cls2 = (MyClass)cls1.clone();  
        System.out.println(cls2);  
  
        System.out.println(cls1==cls2);  
        System.out.println(cls1.getValue() == cls2.getValue());  
        System.out.println(cls1.getValues() == cls2.getValues());  
  
        cls1.addInteger(25, 26, 27);  
        System.out.println(cls1);  
        System.out.println(cls2);  
    }  
}
```

12 [15, 17, 10]	12 [15, 17, 10]
false	true
cls1.setValue(37);	true
System.out.println(cls1);	true
System.out.println(cls2);	12 [15, 17, 10, 25, 26, 27]
	12 [15, 17, 10, 25, 26, 27]
	37 [15, 17, 10, 25, 26, 27]
	12 [15, 17, 10, 25, 26, 27]

prints

Dr. Cristina Marinescu 16

protected Object clone() throws CloneNotSupportedException from class Object

```
class MyClass implements Cloneable {  
    private int myValue;  
    private ArrayList<Integer> myValues = new ArrayList<Integer>();  
  
    public void setValue(int myValue) { this.myValue = myValue; }  
  
    public void addInteger(Integer... ints) {  
        for(Integer i : ints) myValues.add(i);  
    }  
  
    public String toString() {  
        return myValue + " " + myValues.toString();  
    }  
  
    public ArrayList<Integer> getValues() { return myValues; }  
  
    public int getValue() { return myValue; }  
  
    protected Object clone() throws CloneNotSupportedException {  
        MyClass myClone = (MyClass)super.clone();  
        myClone.myValues = (ArrayList<Integer>)myValues.clone();  
        return myClone;  
    }  
}
```

Deep
Cloning

Dr. Cristina Marinescu 17

Deep Cloning

```
public class CloningSample {  
    public static void main(String[] args) throws CloneNotSupportedException {  
        MyClass cls1 = new MyClass();  
        cls1.setValue(12);  
        cls1.addInteger(15, 17, 10);  
        System.out.println(cls1);  
  
        MyClass cls2 = (MyClass)cls1.clone();  
        System.out.println(cls2);  
  
        System.out.println(cls1==cls2);  
        System.out.println(cls1.getValue() == cls2.getValue());  
        System.out.println(cls1.getValues() == cls2.getValues());  
  
        cls1.addInteger(25, 26, 27);  
        System.out.println(cls1);  
        System.out.println(cls2);  
    }  
}
```

12 [15, 17, 10]	12 [15, 17, 10]
false	true
cls1.setValue(37);	false
System.out.println(cls1);	12 [15, 17, 10, 25, 26, 27]
System.out.println(cls2);	12 [15, 17, 10]
	37 [15, 17, 10, 25, 26, 27]
	12 [15, 17, 10]

prints

Dr. Cristina Marinescu 18

Class LinkedHashSet

```
[defines the iteration ordering =  
the order in which the elements were inserted]  
  
public class MyElementsLinkedSet {  
    public static void main(String[] args) {  
        MyElement m1 = new MyElement(3);  
        MyElement m2 = new MyElement(4);  
        MyElement m3 = new MyElement(1);  
  
        HashSet<MyElement> set = new HashSet<MyElement>();  
        set.add(m1);  
        set.add(m2);  
        set.add(m3);  
        System.out.println(set);  
  
        LinkedHashSet<MyElement> linkedSet = new LinkedHashSet<MyElement>();  
        linkedSet.add(m1);  
        linkedSet.add(m2);  
        linkedSet.add(m3);  
        System.out.println(linkedSet);  
  
        MyElement m4 = new MyElement(2);  
        set.add(m4); linkedSet.add(m4);  
        System.out.println(set); System.out.println(linkedSet);  
    }  
}
```

Dr. Cristina Marinescu

19

```
class CompareExample {  
    public static void main(String[] args){
```

```
        String s1 = new String("ABS");  
        String s2 = new String("BSA");  
        System.out.println(s1.compareTo(s2));  
        System.out.println(s2.compareTo(s1));
```

```
        s1 = new String("ABS");  
        s2 = new String("AB");  
        System.out.println(s1.compareTo(s2));  
        System.out.println(s2.compareTo(s1));
```

```
        s1 = new String("ABS");  
        s2 = new String("AC");  
        System.out.println(s1.compareTo(s2));  
        System.out.println(s2.compareTo(s1));  
    }
```

-1

1

-1

1

-1

1

Dr. Cristina Marinescu

20

Class TreeSet

The elements are ordered using their natural ordering, or by a Comparator provided at set creation time, depending on which constructor is used.

```
public class MyStringElement implements Comparable<MyStringElement> {  
    private String myString;  
  
    public MyStringElement(String myString) {  
        this.myString = myString;  
    }  
  
    public String getString() {  
        return myString;  
    }  
  
    public String toString() {  
        return myString;  
    }  
  
    public int compareTo(MyStringElement o) {  
        return myString.compareTo(((MyStringElement)o).myString);  
    }  
}
```

Dr. Cristina Marinescu

21

```
public class MyElementsTreeSet1 {  
    public static void main(String[] args){
```

```
        MyStringElement m1 = new MyStringElement("3");  
        MyStringElement m2 = new MyStringElement("4");  
        MyStringElement m3 = new MyStringElement("1");
```

```
        TreeSet<MyStringElement> tree = new TreeSet<MyStringElement>();  
        tree.add(m1);  
        tree.add(m2);  
        tree.add(m3);  
        tree.add(new MyStringElement("1"));  
        System.out.println(tree);  
    }
```

Dr. Cristina Marinescu

22

[1, 3, 4]

```

public class MyElementsTreeSet2 {
    public static void main(String[] args){
        MyStringElement m1 = new MyStringElement("3");
        MyStringElement m2 = new MyStringElement("4");
        MyStringElement m3 = new MyStringElement("1");

        TreeSet<MyStringElement> tree = new TreeSet<MyStringElement>(
            new Comparator<MyStringElement>() {
                public int compare(MyStringElement o1, MyStringElement o2)
                {
                    return o1.getString().compareTo(o2.getString())*(-1);
                }
            });

        tree.add(m1);
        tree.add(m2);
        tree.add(m3);
        System.out.println(tree);
    }
}

```

Dr. Cristina Marinescu 23

Variable number of arguments

```

import java.util.TreeSet;

public class MyElementsTreeSet3 {
    private TreeSet<MyStringElement> set = new TreeSet<MyStringElement>();

    public void add(MyStringElement...elements){
        for(MyStringElement e : elements)
            set.add(e);
    }

    public String toString() {
        return set.toString();
    }

    public static void main(String[] args) {
        MyElementsTreeSet3 set = new MyElementsTreeSet3();

        MyStringElement m1 = new MyStringElement("3");
        MyStringElement m2 = new MyStringElement("4");
        MyStringElement m3 = new MyStringElement("1");

        set.add();
        System.out.println(set);
        set.add(m1);
        System.out.println(set);
        set.add(m1, m2, m3);
        System.out.println(set);
    }
}

```

Dr. Cristina Marinescu 24

Class HashMap

[This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.]

```

import java.util.HashMap;

public class JavaHashMap {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<String, Integer>();
        Integer count;

        count = map.get("string1");
        map.put("string1", count == null ? 1 : count + 1);

        count = map.get("string1");
        map.put("string1", count == null ? 1 : count + 1);

        count = map.get("string7");
        map.put("string7", count == null ? 1 : count + 1);

        count = map.get("string2");
        map.put("string2", count == null ? 1 : count + 1);

        System.out.println(map);
    }
}

```

Dr. Cristina Marinescu 25

**for (String key: map.keySet())
System.out.println(key);**

**for (Integer value: map.values())
System.out.println(value);**

fore more details please consult

<http://docs.oracle.com/javase/7/docs/api/>

Dr. Cristina Marinescu 26

Class TreeMap

[The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.]

```
public class JavaTreeMap {  
    public static void main(String[] args) {  
        TreeMap<String, Integer> map = new TreeMap<String, Integer>();  
        Integer count;  
  
        count = map.get("string1");  
        map.put("string1", count == null ? 1 : count + 1);  
  
        count = map.get("string1");  
        map.put("string1", count == null ? 1 : count + 1);  
  
        count = map.get("string7");  
        map.put("string7", count == null ? 1 : count + 1);  
  
        count = map.get("string2");  
        map.put("string2", count == null ? 1 : count + 1);  
  
        System.out.println(map);  
    }  
}
```

