Arrays, Strings and Collections

Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory
Dept. of Computer Science and Software Engineering
University of Melbourne, Australia
http://www.buyya.com

1

**toString() Method**

- `toString()` method is a special method that can be defined in any class.
- This method should return a String argument.
- When an object is used in a String concatenation operation or when specified in print statement, this method gets invoked automatically.

2

**toString() Method - Example**

class Circle {
    double x, y, r;
    public Circle (double centreX, double centreY, double radius) {
        x = centreX; y = centreY; r = radius;
    }
    public String toString()
    {
        String s = "I am a Circle with centre [" + x + "," + y + "]
        and radius [" + r + "]";
        return s;
    }
}

3

**StringBufferClass**

- Unlike the String class, StringBuffer class is mutable (changeable).
- Use StringBuffer class in operations where the string has to be modified.

4

**StringBuffer class - Constructors**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public StringBuffer()</td>
<td>Constructs a StringBuffer with an empty string.</td>
</tr>
<tr>
<td>public StringBuffer(String str)</td>
<td>Constructs a StringBuffer with initial value of str.</td>
</tr>
</tbody>
</table>
StringBuffer class – Some operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public int length()</code></td>
<td>Returns the length of the buffer</td>
</tr>
<tr>
<td><code>public synchronized void setCharAt(int index, char ch)</code></td>
<td>Replaces the character at the specified position</td>
</tr>
<tr>
<td><code>s1.setLength(int n)</code></td>
<td>Truncates or extends the buffer. If (n&lt;s1.length()), s1 is truncated; else zeros are added to s1</td>
</tr>
<tr>
<td><code>public StringBuffer append(String str)</code></td>
<td>Appends the string to this string buffer</td>
</tr>
<tr>
<td><code>public StringBuffer append(int i)</code></td>
<td>Append of other data items (float, char, etc.) is supported</td>
</tr>
</tbody>
</table>

Inserting a String in Middle of Existing StringBuffer

- StringBuffer str = new StringBuffer("Object Language");
- String aString = new String(str.toString());
- Int pos = aString.indexOf(" Language");
- str.insert(pos, " Oriented ");
- What will out put of at this point:
  - System.out.println("Modified String:" + str);
- What will be string after executing (modifying character):
  - str.setChar(6,'-');

StringTokenizer

- Breaks string into parts, using delimiters.
- The sequence of broken parts are the tokens of the string.
- More than one delimiter can be specified.
- The tokens can be extracted with or without the delimiters.

StringTokenizer - Functionality

- Consider the following String
  - CREATE_USER:1234567;John;Smith
- Separate the tokens
  - CREATE_USER
  - 1234567
  - John
  - Smith

StringTokenizer - Constructors

- `public StringTokenizer(String str, String delim, boolean returnTokens)`
  - Creates a StringTokenizer with the specified delimiter. If returnTokens is true the delimiters are also returned.
- `public StringTokenizer(String str, String delim)`
  - Delimiters are not returned
- `public StringTokenizer(String str)`
  - Delimiters are (" \

StringTokenizer - Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public boolean hasMoreTokens()</code></td>
<td>Returns true if more tokens are found</td>
</tr>
<tr>
<td><code>public String nextToken()</code></td>
<td>Returns the next token of the String</td>
</tr>
<tr>
<td><code>public String nextToken(String delim)</code></td>
<td>Switches the delimiter set to characters in delim and returns the next token</td>
</tr>
<tr>
<td><code>public int countTokens()</code></td>
<td>Returns the count of remaining tokens</td>
</tr>
</tbody>
</table>
StringTokenizer - example

```java
import java.util.StringTokenizer;
class TokenizerExample {
    public static void main(String[] args)
    {
        String str = "CREATE_USER:123456;John;Smith";
        StringTokenizer tokens = new StringTokenizer(str, ":;");
        while (tokens.hasMoreTokens())
            System.out.println(tokens.nextToken());
    }
}
```

Collections

- **Arrays** are used to hold groups of *specific* type of items
- **Collections (container)** designed to hold *generic* (any) type of objects
- Collections let you store, organize and access objects in an efficient manner.

Legacy Collection Types

- Vector
- Stack
- Dictionary
- HashTable
- Properties
- Enumeration

Vector

- The Vector class implements a growable array of objects.
- Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
- In Java this is supported by Vector class contained in `java.util` package. The Vector class can be used to create generic dynamic arrays that hold objects of any type or any number. The objects do not have to be homogeneous.
- Like arrays, Vectors are created as follows:
  - Vector list = new Vector(); // declaring without size
  - Vector list = new Vector(3); // declaring with size

Vector properties

- Vectors possess a number of advantages over arrays:
  - It is convenient to use vectors to store objects.
  - A vector can be used to store list of objects that may vary in size.
  - We can add and delete objects from the list as an when required.
- But vectors cannot be used to store basic data types (int, float, etc.); we can only store objects. To store basic data type items, we need convert them to objects using “wrapper classes” (discussed later).

Important Methods in Vector class

- addElement(Object item)
- insertElementAt(Object item, int index)
- elementAt(int index) – get element at index
- removeElementAt(int index)
- size() –
- clone() - Returns a clone of this vector.
- clear() - Removes all the elements from this Vector.
- get(int index) - Returns the element at the specified position in this Vector.
- copyInto(array) — copy all items from vector to array.
Vector – Example 1

```java
import java.util.*;
public class VectorOne {
    public static void main(String[] args) {
        Vector circleVector = new Vector();
        System.out.println("Vector Length = ", circleVector.size()); // 0
        for (int i = 0; i < 5; i++) {
            circleVector.addElement(new Circle(i)); // radius of the Circles 0,1,2,3,4
        }
        System.out.println("Vector Length = " + circleVector.size()); // 5
    }
}
```

Vector – Example 2

```java
import java.util.*;
public class VectorTwo {
    public static void main(String[] args) {
        // code from VectorOne goes here
        circleVector.insertElementAt(new Circle(20), 3);
        System.out.println("Vector Length = "+ circleVector.size()); // 6
        for (int i = 0; i < 6; i++) {
            System.out.print("Radius of element "+ i + "] = " + Circle.circleVector.elementAt(i).getRadius());
        }
        // radius of the Circles are 0,1,2,20,3,4
    }
}
```

Hash Table (Hashtable class)

- Allows associating values with keys.
- Allows efficient look ups for the value associated with the key.
- This class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value.
- Useful Operations:
  - put(Object key, Object value);
  - remove(Object key);
  - get(Object key);

HashTable put() / get() operations

- The following example creates a hashtable of numbers. It uses the names of the numbers as keys:
  - Hashtable numbers = new Hashtable();
  - numbers.put("one", new Integer(1));
  - numbers.put("two", new Integer(2));
  - numbers.put("three", new Integer(3));
- To retrieve a number, use the following code:
  - Integer n = (Integer)numbers.get("two");
  - if (n != null) { System.out.println("two = " + n); }

HashTable - Example

```java
import java.util.*;
public class HashtableDemo {
    public static void main(String[] args) {
        Hashtable tbl = new Hashtable();
        Student s, sRet;
        s = new Student("121212", "John");
        tbl.put(s.getId(), s);
        s = new Student("100000", "James");
        tbl.put(s.getId(), s);
        sRet = (Student) tbl.get("121212");
        System.out.println("Student name is = " + sRet.getName());
        // Student name is = John
    }
}
```

Enumeration

- Used to enumerate or iterate through a set of values in a collection.
- Useful for iterating Hashtables – no index.
- Useful Operations:
  - hasMoreElements();
  - nextElement();
### Enumeration - Example

```java
import java.util.*;
public class EnumerationDemo{
    public static void main(String[] args) {
        Hashtable tbl = new Hashtable();
        Student sRet;
        s = new Student("121212", "John");
        tbl.put(s.get陔(), s);
        s = new Student("100000", "James");
        tbl.put(s.get陔(), s);
        Enumeration e = tbl.elements();
        while ( e.hasMoreElements() ) {
            sRet = (Student) e.nextElement();
            System.out.println("Student name is = " + sRet.getName());
            // Student name is = James
            // Student name is = John
        }
    }
}
```

### Wrapper Classes

- As pointed out earlier, collections cannot handle basic data types such as int, float. They can converted into object types by using the wrapper classes supported by java.lang package.

<table>
<thead>
<tr>
<th>Basic Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
</tbody>
</table>

### Methods in Wrapper Classes

- **Constructors:**
  - `Integer intVal = new Integer(i);`
  - `Float floatVal = new Float(f);`
- **Converting objects to basic values**
  - `int i = intVal.intValue();`
  - `float f = floatValue.floatValue();`
- **Converting Numbers to Strings**
  - `str = Integer.toString(i)`
  - `str = Float.toStrin(f);`

### Methods in Wrapper Classes

- **String Objects to Numeric Objects**
  - `Integer intVal = Integer.valueOf(str);`
  - `Float floatVal = Float.valueOf(str);`
- **Numeric Strings to Basic Types**
  - `int i = Integer.parseInt(str);`
  - `long l = Long.parseLong(str);`
  - These methods throw exception (NumberFormatException) if the value of the str does represent an integer. Exception are a OO way of reporting errors. More on it later.

### Summary

- A special method, `toString()`, can be defined in any Java class, which gets invoked when one tries to concatenation operation with Strings.
- Collections are like arrays, but can hold any objects, dynamically expandable, and supports their easy manipulation.
- Java has strong support for Collections, which are very useful when developing large-scale software development.
- Wrapper classes helps in manipulating basic data types as Objects.