

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa
Melksham
Wiltshire
UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Strings

Java supports a Unicode character set that allows for more than 65,000 different characters. Single characters can be held in char variables, and multiple character sequences (strings) can be held in char arrays, StringBuffer objects and String objects.

<i>Character variables</i>	4
<i>String constants</i>	6
<i>Creating String objects</i>	8
<i>Operations on strings</i>	13
<i>Comparing strings</i>	14
<i>Accessing characters within strings</i>	17
<i>Character arrays v String objects</i>	21
<i>String buffers</i>	21

With our study of classes and objects we have seen how data is at the centre of a Java program, and how that data is accessed using methods within classes.

We have seen how data is held within a class and how that data can refer to all class members, to an individual class member, or be just a one-off working piece of information.

Our basic building blocks have been integers and floating point numbers.

But look back at our card decks and compare them with how we would really play cards. Would we ask

"Have you card number 48?" or

"Have you the Queen of Hearts?"?

2.1 Character variables

We need to be able to handle characters as well as numbers. We *could* use a data type **char** for character ...

```
// Well House Consultants2004.
import java.lang.Math;
/** Character Variable*/
public class ramsbury
{
    public static void main(String[] args)
    {

        int card = 1 + (int) (Math.random()*52.0);
        System.out.print("You have drawn ");
        System.out.println(card);

        int rank = (card-1) / 4;
        int suit = (card-1) % 4;

        char [] suites = { 'C','D','H','S' };
        char [] ranks = { '2','3','4','5','6','7','8',
                        '9','T','J','Q','K','A' };

        char rank_letter = ranks[rank];
        char suit_letter = suites[suit];

        System.out.print("You have drawn ");
        System.out.print(rank_letter);
        System.out.print(" of ");
        System.out.println(suit_letter);

    }
}
```

Figure 1 Running public class ramsbury

```
seal% java ramsbury
You have drawn 8
You have drawn 3 of S
seal% java ramsbury
You have drawn 40
You have drawn J of S
seal% java ramsbury
You have drawn 29
You have drawn 9 of C
seal%
```

This lets us translate our card numbers in the pack into the names of the cards.

Notice data type **char** being used in a similar way to other data types - single variables and arrays.

char constants are written with the character between single quotes.

chars may be compared in just the same way as **float** or **int** variables. Let's extend our draw to include a load of Jokers.

```
// Well House Consultants 2004.
import java.lang.Math;
/** Character Variable*/
public class highworth
{
    public static void main(String[] args)
    {

        int card = (1 + (int) (Math.random()*52.0))
            * (int) (Math.random()*2.0) ;
        // add 52 jokers!!
        System.out.print("You have drawn ");
        System.out.println(card);

        int rank = (card-1) / 4;
        int suit = (card-1) % 4;

        char [] suites = {'C','D','H','S'};
        char [] ranks = {'2','3','4','5','6','7','8',
            '9','T','J','Q','K','A'};
```

Figure 2 Running public class highworth

```
seal% java highworth
You have drawn 8
You have drawn 3 of S
seal% java highworth
You have drawn 0
You have drawn **
seal% java highworth
You have drawn 0
You have drawn **
seal% java highworth
You have drawn 41
You have drawn Q of C
seal%
```

The only change in the above section is the algorithm to determine which card is selected. It now selects a random number in the range 1 to 52, but then multiplies the result by zero in half the cases. (By the way, we have added this many Jokers for training purposes so that you can quickly see the program in action!)

```
char rank_letter, suit_letter;
if (card == 0) {
    rank_letter = '*';
    suit_letter = '*';
} else {
    rank_letter = ranks[rank];
    suit_letter = suites[suit];
}
System.out.print("You have drawn ");
```

```

    System.out.print(rank_letter);
    if (rank_letter != '*')
        System.out.print(" of ");
    System.out.println(suit_letter);
}
}

```

Remember:

The `rank_letter` and `suit_letter` variables are **declared** before the `if` statement, otherwise they would have a scope only within the `if` or `else` clauses.

`!=` is used as the "not equals" comparison.

If there is no `{}` block after an `if` statement, the `if` applies up to the following `;` character.

2.2 String constants

OK, it works. But perhaps it would be more descriptive if the program printed "Queen of Clubs" rather than "Q of C".

This requires whole sequences of characters for which we can use an object of type **String**.

Let's go back to the `ramsbury` example and extend it to use **String** objects.

```

// Well House Consultants2004.
import java.lang.Math;

/** string object*/

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

        int card = 1 + (int) (Math.random()*52.0);
        System.out.print("You have drawn ");
        System.out.println(card);

        int rank = (card-1) / 4;
        int suit = (card-1) % 4;

        String [] suites = { "Clubs","Diamonds",
                            "Hearts","Spades"};
        String [] ranks = { "2","3","4","5",
                            "6","7","8",
                            "9","10","Jack",
                            "Queen","King","Ace"};

        String rank_name = ranks[rank];
        String suit_name = suites[suit];

        System.out.print("You have drawn ");
        System.out.print(rank_name);
        System.out.print(" of ");
        System.out.println(suit_name);
    }
}

```

Figure 3 Running public class marston

```

seal% java marston
You have drawn 40
You have drawn Jack of Spades
seal% java marston
You have drawn 47
You have drawn King of Hearts
seal% java marston
You have drawn 3
You have drawn 2 of Hearts

```

String constants¹ can be multiple characters in length and are written using double quotes.

They can contain as few or as many characters as you like:

9	1 character
10	2 characters
Queen	5 characters

All these examples contain just letters and digits, but String constants may contain any characters, including spaces, control codes, etc. However, if you want to include other special characters, you may have a problem.

```
String demo = "Don't do it" he said";
```

This would not work. Which double quotes delimit the string and which are part of the string?

Special characters may be included in Strings using an escape sequence starting with a `\` character.

Two-character sequences include:

<code>\n</code>	new line
<code>\t</code>	tab
<code>\"</code>	double quote
<code>\\</code>	backslash
<code>\b</code>	backspace

You can also use unicodes to represent any character. Unicodes are how the strings are held internally:

<code>\u0068</code>	lower case h
<code>\u00a3</code>	£ sign

Here's a code chunk showing these codes in action:

```
String [] suites = { "\"Hit \u0068im\"\\nhe said",
                    "\"worth \u00a3s\"",
                    "Hearts", "Spades"};
```

```

seal% java blunsdon
You have drawn 18
You have drawn 6 of "worth £s"
seal%

```

Individual **char** constants may also be written using these escape sequences or unicodes.

¹ you have noticed that we're using a capital S, haven't you?

2.3 Creating String objects

Strings are objects rather than simple data types.¹ They can be created and assigned just like other objects:

```
String rank_name =  
new String("Initial demo rank string\n");
```

(Note: the constructor for type string can take a value to assign to the string.)

This notation can be shortened:

```
String suit_name = "Another String demo\n";
```

String objects can be created without any initial value:

```
String yet = new String();
```

or simply

```
String moresome;
```

Arrays of String objects can also be created with an initial set of values:

```
String [] ranks = { "2", "3", "4", "5",  
"6", "7", "8",  
"9", "10", "Jack",  
"Queen", "King", "Ace"};
```

or without any values:

```
String [] another = new String[10];
```

It will be important later to remember that String names are instance variables and conform to the same rules.

It is perfectly valid to use a statement such as:

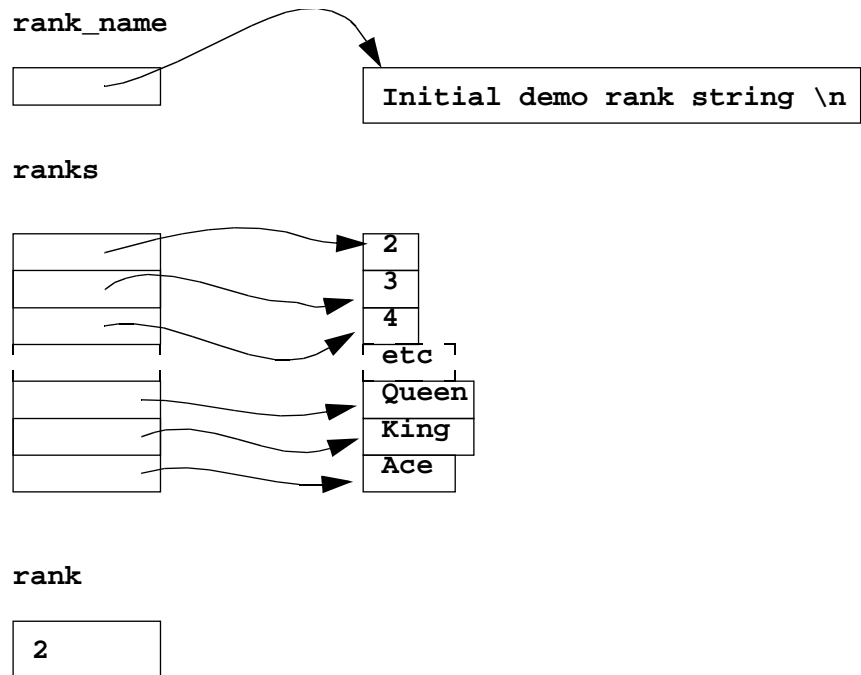
```
rank_name = ranks[rank];
```

but remember that this does not copy the string held in the ranks array. Rather, it causes the instance variable *rank_name* to become an alternative name for an instance variable from the ranks array. The previous instance that *rank_name* referred to may be lost unless it has some other instance variable name.

Let's run through that again.

¹ that's why we use a capital S

Figure 4 Diagram showing variable usage prior to execution of:
`rank_name =`
`ranks[rank];`



Here are the pertinent lines of the code:

```

    int rank = (card-1) / 4;
// more code
    String rank_name =
        new String("Initial demo rank string\n");
// more code
    String [] ranks = { "2","3","4","5",
                       "6","7","8",
                       "9","10","Jack",
                       "Queen","King","Ace"};
// more code
    rank_name = ranks[rank];

```

`rank_name` is an instance variable

`ranks` is an array of instance variables

`rank` is an integer variable

When we execute the assignment statement:

```
rank_name = ranks[rank];
```

the diagram changes as shown.

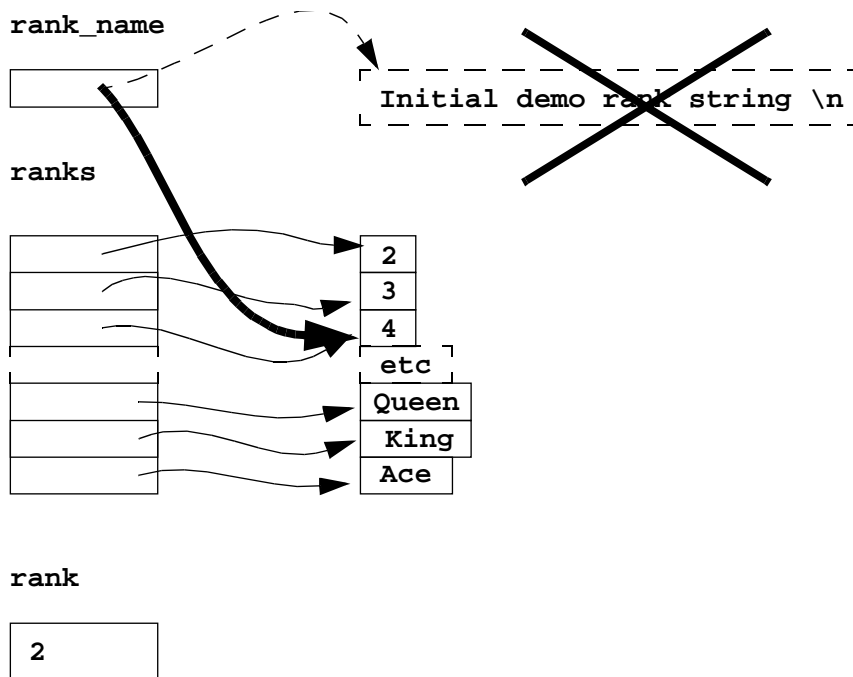


Figure 5 Diagram showing variable usage as `rank_name = ranks[rank];` is executed

- The instance variable `rank_name` now refers to element number 2 in the `ranks` array of strings (element number 2 contains the string "4" in this example)
- The character string "Initial demo rank string \n" is no longer referenced from anywhere, and is automatically deleted

Can you work out what the following assignments do?

```
yet = suit_name = suites[suit];
another[4] = rank_name;
moresome = another[4];
```

Recall how these variables were declared:

```
String yet = new String();
String suit_name = "Another String demo\n";
String [] suites = { "Clubs", "Diamonds",
                    "Hearts", "Spades" };
String rank_name =
    new String("Initial demo rank string\n");
rank_name = ranks[rank];
String [] another = new String[10];
String moresome;
```

Yes, more manipulation of instance variables. Let's combine all these examples into a training program so that you can revise and see the complete picture:

```
// Well House Consultants2004.
import java.lang.Math;
/** string instances*/
public class cricklade
{
    public static void main(String[] args)
    {

        int card = 1 + (int) (Math.random()*52.0);
        System.out.print("You have drawn ");
        System.out.println(card);

        int rank = (card-1) / 4;
        int suit = (card-1) % 4;
```

```
String rank_name =
    new String("Initial demo rank string\n");
System.out.print(rank_name);

String suit_name = "Another String demo\n";
System.out.print(suit_name);
String [] ranks = { "2","3","4","5",
                   "6","7","8",
                   "9","10","Jack",
                   "Queen","King","Ace"};

String [] suites = { "Clubs","Diamonds",
                    "Hearts","Spades"};

String yet = new String();
String moresome;
String [] another = new String[10];

rank_name = ranks[rank];
yet = suit_name = suites[suit];
another[4] = rank_name;
moresome = another[4];

System.out.print("You have drawn ");
System.out.print(rank_name);
System.out.print(" of ");
System.out.println(suit_name);

System.out.print("Let's confirm that:
                the ");
System.out.print(ranks[rank]);
System.out.print(" of ");
System.out.println(suites[suit]);

System.out.print
    ("And to be utter sure: the ");
System.out.print(moresome);
System.out.print(" of ");
System.out.println(yet);
    }
}
```

```

seal% java cricklade
You have drawn 41
Initial demo rank string
Another String demo
You have drawn Queen of Clubs
Let's confirm that: the Queen of Clubs
And to be utter sure: the Queen of Clubs
seal% java cricklade
You have drawn 18
Initial demo rank string
Another String demo
You have drawn 6 of Diamonds
Let's confirm that: the 6 of Diamonds
And to be utter sure: the 6 of Diamonds
seal%

```

Figure 6 Running public class cricklade

2.4 Operations on strings

Strings are objects, remember, so that operations on strings are performed using methods. Some of the methods are achieved by overloading operators.

What do we want to do with strings?

Concatenate them ... add one string onto the end of another:

```

String rank_name = ranks[rank];
String suit_name = suites[suit];
String card_name = rank_name + " of " + suit_name;

System.out.print("You have drawn ");
System.out.println(card_name);

```

```

seal% java lechlade
You have drawn 48
You have drawn King of Spades
seal%

```

Figure 7 Running public class lechlade

If the + operator is used on a mixture of numeric variables and strings, it will convert to character strings as necessary and concatenate.

```

// Well House Consultants©2004.

/** string concatenation*/

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

        String demo_one;
        int demo_two;

        demo_one = "" + 10 + 10;
        demo_two = 10 + 10;

        System.out.println

```

```

        ("demo_one has the value " + demo_one);
    System.out.println
        ("demo_two has the value " + demo_two);
    }
}

```

Figure 8 Running public class *inglesham*

```

seal% java inglesham
demo_one has the value 1010
demo_two has the value 20
seal%

```

2.5 Comparing strings

We might also want to compare strings.

Let's see a code snippet which sets up two strings that we want to compare with each other:

```

String affirm = "yes";
System.out.print
    ("Please enter yes or no : ");
String you_said = WellHouseInput.readLine();

```

(The *WellHouseInput.readLine* method is provided as a quick way of reading user input for this course.)

You might think that the following code would test whether the string entered is the word "yes" ...

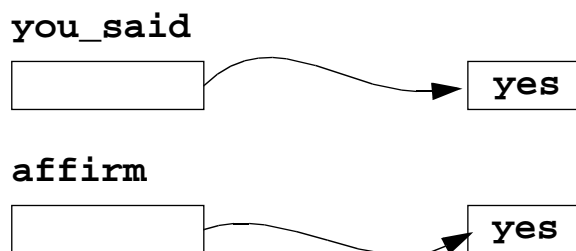
```

if (you_said == affirm) {
    System.out.println("GOT WHAT I WANTED");
} else {
    System.out.println("Oh dear ....");
}

```

but it doesn't. What it actually tests is whether *you_said* and *affirm* both point to the same instance on an object of the type **String**.

Two instances of type **String**:



Although both instances contain identical data using `==`, to compare them will give a *false* result.

Only in this situation ...

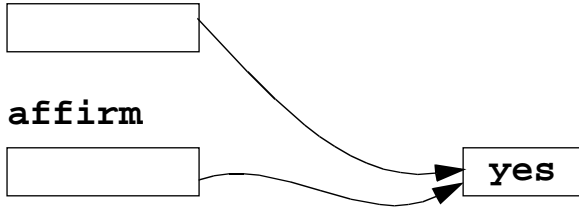
you_said



affirm



yes



will `==` give a *true* result.

If you wish to compare two different String objects to see if the contents are the same, use the equal method from one of the objects, giving the other object as a parameter:

```
if (you_said.equals(affirm)) {
    System.out.println("GOT WHAT I WANTED");
}
```

Let's confirm that in a complete example:

```
// Well House Consultants 2004.
/** string comparison*/
public class ashton
{
    public static void main(String[] args)
    {
        String affirm = "yes";
        System.out.print
            ("Please enter yes or no : ");
        String you_said = WellHouseInput.readLine();
        System.out.println("you said " + you_said);
        System.out.println("hoping for " + affirm);

        // you may guess the following will work but
        // it will NOT.
        System.out.print("Trying \"==\" ... ");
        if (you_said == affirm) {
            System.out.println("GOT WHAT I WANTED");
        } else {
            System.out.println("Oh dear ....");
        }

        // But this WILL do what you hope for ...
        System.out.print("Trying \"equals\" ... ");
        if (you_said.equals(affirm)) {
            System.out.println("GOT WHAT I WANTED");
        } else {
            System.out.println("Oh dear ....");
        }
    }
}
```

Figure 9 Running public class ashton

```

seal% java ashton
Please enter yes or no no
you said no
hoping for yes
Trying "==" ... Oh dear ....
Trying "equals" ... Oh dear ....
seal% java ashton
Please enter yes or no yes
you said yes
hoping for yes
Trying "==" ... Oh dear ....
Trying "equals" ... GOT WHAT I WANTED
seal%

```

- If you want to check that your user entered the word "yes" but you don't care about the case, you should use the `equalsIgnoreCase` method.
- If you want to check just the start of the string, the `startsWith` method can be used, and to check just the end of a string, use the `endsWith` method.
- In each case, the argument you give to the method could of course be a String constant, a String object, or an expression which evaluates to a String object.

```

// Well House Consultants2004.

/** string comparisons*/

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

        System.out.print
            ("Please enter a fruit : ");
        String you_said = WellHouseInput.readLine();

        System.out.println("you said " + you_said);

        if (you_said.equals("orange"))
            System.out.println("Mmmm - Florida Orange Juice!");

        if (you_said.equalsIgnoreCase("orange"))
            System.out.println("Orange Juice!");

        if (you_said.startsWith("o"))
            System.out.println("starts like an orange");

        if (you_said.endsWith("ge"))
            System.out.println("ends like an orange");

    }
}

```

Figure 10 Running public class parton

```

seal% java parton
Please enter a fruit orange
you said orange
Mmm - Florida Orange Juice!
Orange Juice!
starts like an orange
ends like an orange
seal% java parton
Please enter a fruit Orange
you said Orange
Orange Juice!
ends like an orange
seal% java parton
Please enter a fruit Change
you said Change
ends like an orange
seal%

```

If you want to check which of the two strings comes first in the alphabet, use the `compareTo` method.

It returns an integer: negative (object on which it is called is first alphabetically); positive (argument is first); or zero (they're the same).

To sort a whole load of strings, such a method would be called repeatedly.

2.6 Accessing characters within strings

The `startsWith` method can take a second (integer) parameter to start checking from somewhere other than the first character.

Of course, you'll want to know other things about strings as well, such as:

- What is the length of a string?
- Where is the first/next/last "x" in a string?

For the length, there is a `length` method.

For the first occurrence, use `indexOf` with a single parameter, and for the next occurrence use `indexOf` with two parameters.

Working backwards, use the `lastIndexOf` method instead.

Both `indexOf` and `lastIndexOf` will accept either a `char` or a `String` object as the search pattern.

```

// Well House Consultants2004.

/** string analysis*/

public class minety
{
    public static void main(String[] args)
    {
        System.out.print
            ("Please enter a phrase : ");

        String you_said = WellHouseInput.readLine();

        System.out.println("you said " + you_said);
        System.out.println("length of data " +
            you_said.length());

        int posn=-1;

```

```

        while ((posn = you_said.indexOf(' ',posn+1))>=0)
            System.out.println ("Space at position "+posn);
    }
}

```

Figure 11 Running public class minety

```

seal% java minety
Please enter a phrase This is a course
you said This is a course
length of data 16
Space at position 4
Space at position 7
Space at position 9
seal%

```

Remember: arrays and positions start at zero, not one!

Individual characters may be extracted by using the **charAt** method, and substrings by using the **substring** method.

```

// Well House Consultants©2004.

/** string analysis*/

public class oaksey
{
    public static void main(String[] args)
    {
        System.out.print
            ("Please enter a phrase : ");
        String you_said = WellHouseInput.readLine();

        int old_posn=0,posn=0;
        while (posn>=0)
        {
            posn = you_said.indexOf(' ',old_posn);
            String next_word = (posn>0) ?
                you_said.substring(old_posn,posn):
                you_said.substring(old_posn);
            System.out.println("Next word: "+next_word);
            old_posn = posn + 1;
        }
    }
}

```

Figure 12 Running public class oaksey

```

seal% java oaksey
Please enter a phrase This is a course
Next word: This
Next word: is
Next word: a
Next word: course
seal%

```

Note that **?** and **:** are alternative forms of the **if** statement.

Also note that **substring** is overloaded.

With two integer parameters, you select the first character you want in the

substring, then the first character beyond the substring:

```
you_said.substring(5,8);
```

gives characters 5 through 7.

With a single integer parameter, you select the first character you want in the substring, which then includes all remaining characters in the incoming string object.

String objects cannot be modified in place.

You can alter strings by using methods like **substring** and **+** to break down and rebuild, but this is inefficient.

You can also use **replace** and **trim** methods, which will return you a new string.

replace Replaces all occurrences of one character with another

trim Removes leading and trailing spaces

Yes, you can write:

```
mystring = mystring.trim();
```

but remember that the old string object will be replaced by the new one. Fine for tidying up a user's input, but pretty inefficient if you have megabytes of data to handle!

There are many other methods available to you for use on Strings - far too many to go into detail here. The `javap` utility looks inside a class file (including ones supplied with Java itself) and will give you a summary of methods and access variables contained therein.

```
[localhost:~/mar03] graham% javap java.lang.String
Compiled from String.java
public final class java.lang.String extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable {
    public static final java.util.Comparator CASE_INSENSITIVE_ORDER;
    public java.lang.String();
    public java.lang.String(java.lang.String);
    public java.lang.String(char[]);
    public java.lang.String(char[],int,int);
    public java.lang.String(byte[],int,int,int);
    public java.lang.String(byte[],int);
    public java.lang.String(byte[],int,int,java.lang.String) throws java.io.UnsupportedEncodingException;
    public java.lang.String(byte[],java.lang.String) throws java.io.UnsupportedEncodingException;
    public java.lang.String(byte[],int,int);
    public java.lang.String(byte[]);
    public java.lang.String(java.lang.StringBuffer);
    java.lang.String(int,int,char[]);
    public int length();
    public char charAt(int);
    public void getChars(int, int, char[], int);
    public void getBytes(int, int, byte[], int);
    public byte getBytes(java.lang.String)[] throws java.io.UnsupportedEncodingException;
    public byte getBytes()[];
    public boolean equals(java.lang.Object);
    public boolean equalsIgnoreCase(java.lang.String);
    public int compareTo(java.lang.String);
    public int compareTo(java.lang.Object);
    public int compareToIgnoreCase(java.lang.String);
    public boolean regionMatches(int, java.lang.String, int, int);
    public boolean regionMatches(boolean, int, java.lang.String, int, int);
    public boolean startsWith(java.lang.String, int);
    public boolean startsWith(java.lang.String);
    public boolean endsWith(java.lang.String);
    public int hashCode();
    public int indexOf(int);
```

```

public int indexOf(int, int);
public int lastIndexOf(int);
public int lastIndexOf(int, int);
public int indexOf(java.lang.String);
public int indexOf(java.lang.String, int);
public int lastIndexOf(java.lang.String);
public int lastIndexOf(java.lang.String, int);
public java.lang.String substring(int);
public java.lang.String substring(int, int);
public java.lang.String concat(java.lang.String);
public java.lang.String replace(char, char);
public java.lang.String toLowerCase(java.util.Locale);
public java.lang.String toLowerCase();
public java.lang.String toUpperCase(java.util.Locale);
public java.lang.String toUpperCase();
public java.lang.String trim();
public java.lang.String toString();
public char toCharArray()[];
public static java.lang.String valueOf(java.lang.Object);
public static java.lang.String valueOf(char[]);
public static java.lang.String valueOf(char[], int, int);
public static java.lang.String copyValueOf(char[], int, int);
public static java.lang.String copyValueOf(char[]);
public static java.lang.String valueOf(boolean);
public static java.lang.String valueOf(char);
public static java.lang.String valueOf(int);
public static java.lang.String valueOf(long);
public static java.lang.String valueOf(float);
public static java.lang.String valueOf(double);
public native java.lang.String intern();
static {};
}

```

As well as **String** objects, you can manipulate sequences of characters by using arrays of characters and by using **StringBuffer** objects. We will look at each of these in turn.

2.7 Character arrays v String objects

If you want to perform low-level manipulation of the characters within a string, you may be advised to use an array of individual characters.

Method **toCharArray** in class **String** will convert a **String** object into an array of characters.

Method **getChars** is more flexible and can be used to extract part of a character string (like **substring** does) and place it starting at any position within a characters array.

Method **copyValueOf** will convert an array of characters into a **String** object.

```

// Well House Consultants2004.
/** string to char array*/

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

        System.out.print
            ("Please enter a phrase : ");
    }
}

```

```

String you_said = WellHouseInput.readLine();

char [] charray = you_said.toCharArray();

for (int k = 1 ; k<charray.length; k++) {
for (int i = 1 ; i<charray.length; i++) {
if (charray[i] < charray [i-1]) {
    char hold = charray[i-1];
    charray[i-1]=charray[i];
    charray[i]=hold;
}}}

String ordered = String.copyValueOf(charray);
System.out.println("Sorting you out, that's "
    +ordered);
}
}

```

```

seal% java crudwell
Please enter a phrase Strings
Sorting you out, that's Sginrst
seal%

```

Figure 13 Running public class crudwell

2.8 String buffers

A **StringBuffer** object should be used when you wish to use a string that can be changed directly.

StringBuffers have many similarities to strings, but do not have the full capabilities of strings in other areas. You'll often see a StringBuffer used as a piece of text is built up, and then the resultant buffer being placed into a **String** for further handling.

Indeed, this is what the compiler itself does with the operator + on strings!

Let's see the creation, extension and copying back to a string in use in a typical piece of code:

```

// Well House Consultants2004.
/** stringBuffer */
public class malmesbury
{
    public static void main(String[] args)
    {
        System.out.print
            ("Please enter a paragraph : ");
        String you_said;

// Create a string buffer, and initialize it empty
        StringBuffer paragraph = new StringBuffer("");

// read from the user until he enters a blank line
// build up entries in the stringbuffer
        while (!(you_said =
            WellHouseInput.readLine()).equals(""))
        {
            paragraph.append(you_said);
            paragraph.append(" ");
        }
    }
}

```

```
// convert back to a string so that we can print re-justified
String textblock = paragraph.toString();

(The rest of the example contains nothing new ... we are just extracting
word-by-word and composing lines of output.)

int old_posn=0, posn=0, line_posn=0;
while (posn>=0)
{
    posn = textblock.indexOf(' ', old_posn);
    String next_word = (posn>0) ?
textblock.substring(old_posn, posn+1):
    textblock.substring(old_posn);
    if ((next_word.length()+line_posn)>30)
    {
        line_posn=0;
        System.out.println("");
    }
    System.out.print(next_word);
    old_posn = posn + 1;
    line_posn+=next_word.length();
}
System.out.println("");
}
}
```

Figure 14 Running public class *malmesbury*

```
seal% java malmesbury
Please enter a paragraph This is a paragraph
of text that will be
rejustified by my Java program

This is a paragraph of text
that will be rejustified by
my Java program
seal%
```

The constructor for the **StringBuffer**:

```
StringBuffer paragraph = new StringBuffer("");
```

may be given ...

- A **String** parameter, which will set the initial string and the initial length
- An **int** parameter, which will set the initial capacity of the string, but no initial value
- No parameter, which will create a String buffer of a default capacity and with no initial value

The **append** method:

```
paragraph.append(you_said);
```

is used to extend the content of the String buffer by adding the contents of the parameter on to the end of the existing content. The parameter may be of many types, and, as well as String, it could be of type **float** or **int** or even of type **boolean** for example.

The **toString** method:

```
String textblock = paragraph.toString();
```

is used to copy the content of a **StringBuffer** into a normal (immutable) **String** object.

You'll note we mentioned the capacity of a StringBuffer. There is no limit on the

number of characters a StringBuffer can hold. The capacity is increased automatically and when necessary.

The length of a StringBuffer is the number of characters it contains, and will be less than or equal to the current capacity.

Current capacity can be found using the **capacity** method, and increased manually using the **ensureCapacity** method.

The current length can be found using the **length** method, and can be altered (longer or shorter) using the **setLength** method.

- If you increase the length of a String buffer, it will be null character filled.
- If you decrease the length of a String buffer, characters will be lost off the end.

There's a rich variety of methods available to you in most of the Java standard classes, and in these training notes we've only got space to tell you about a selection of them. If we tried to provide a full listing, the notes would turn into a reference book.

We suggest that you purchase a good reference book that covers the standard classes in the areas that you'll be using. We ourselves like O'Reilly's "Java in a Nutshell" which includes a single-line reminder of each method in the standard libraries. If you're going on to use the AWT or Swing GUI packages, they're covered in a separate volume, as are the Enterprise classes, and so on. Remember that Java2 1.4 has more than 2,000 classes as standard, and if they have (say) 20 methods each, that's a total of 40,000 methods available to you.

Other methods available on StringBuffers include:

charAt	to return the character at a particular position
getChars	to place a series from a StringBuffer into an array of characters
setCharAt	to set the character at a particular position
Insert	to insert at a position (first parameter) the (second) named parameter

There are lots of methods in lots of classes supplied with Java. You'll need to look up new methods as you get into practical Java programming.

We have given you an overview of StringBuffers and a first example, but you will need to use reference material (the tutor will help you) for exercises that follow, to discover how to use the methods listed (but not fully documented) on the previous pages.

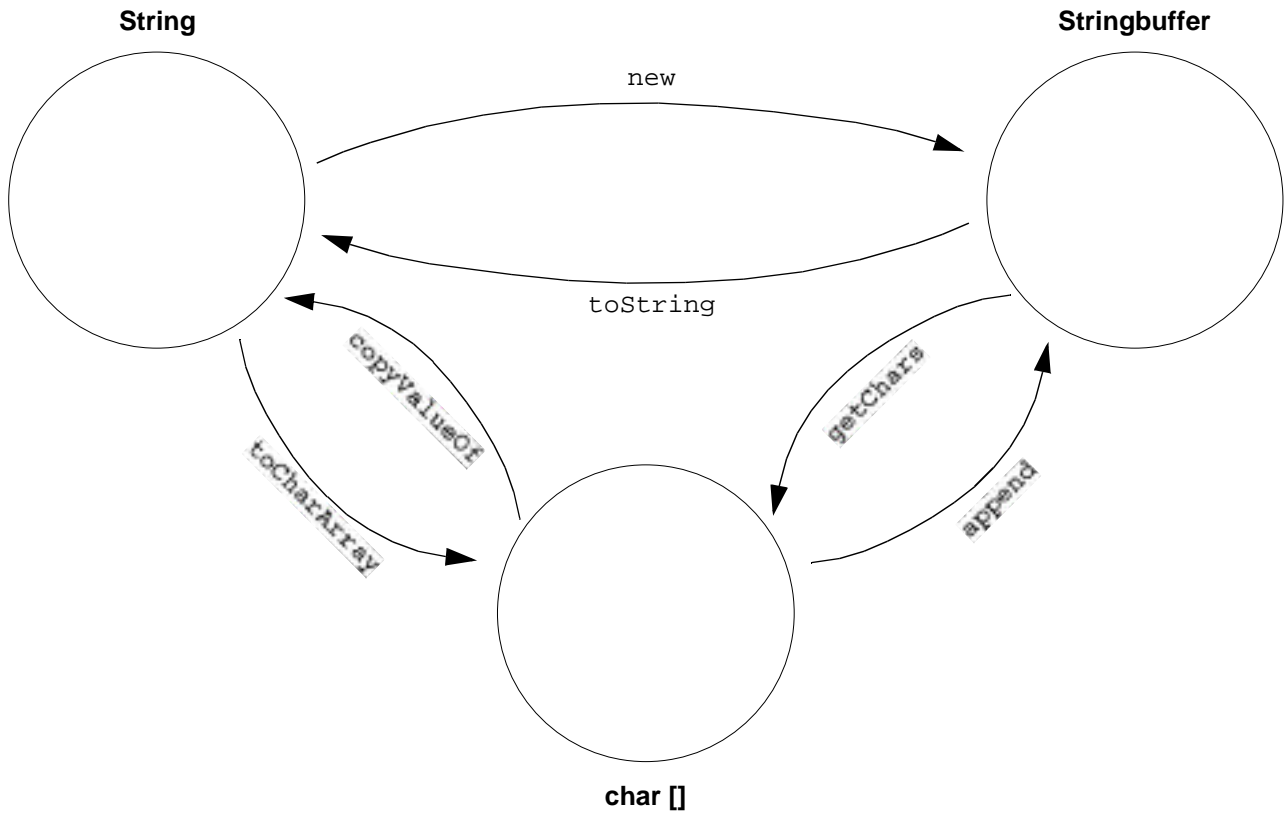


Figure 15 Java uses three different structures to handle groups of characters

Java uses three different structures (Strings, Stringbuffers and char arrays) to handle groups of characters. Each has its strengths and weaknesses, as you have already learnt in this module.

In practice, you might want to use one structure at one time in your program, then move your group of characters to another structure. The diagram above gives you examples of the methods that can be used for each of the conversions.



Exercise

1. Write a class that creates an array of vehicles (or some other type of object that you have already written). Loop through each element of the array, reporting in the number of seats and top speed of every vehicle which is NOT owned by "unknown".
2. When you have the first part of the question working, also print out a comma-separated list of the names of all the vehicle owners. You should build this information up in a StringBuffer while you're looping through all the elements, then print it out at the very end.

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.