JAVA STRINGS

Unlike int variables or double variables, strings are objects. What this implies is that in practice you can do things with strings of text that you can't do with int or double variables. The same applies to the primitive data types: boolean, byte, single, char, float, long and short. They are not objects like strings.

How Java Store Strings

A string is a series of Unicode characters held under a variable name. For example:

```
String someText = "Bill";
```

This tells Java to set up a string object with the four characters "B", "i", "l", and another "l." In the Unicode character set, these values are: `\u0042`, `\u0069`, `\u006c`, `\u006c`. Unicode vales are stored as hexadecimal numbers. Capital letter (A to Z) are stored using the values `\u0041` to `\u005a`, while lowercase letters (a to

z) are stored using the hexadecimal values `\u0061` to `\u007a`.

Because of this system of coding, if you wanted to arrange texts like: "This" and "array", in alphabetic order, you would notice that "This" comes first before 'array'. This happens because lowercase "a" has a hexadecimal value of `\u0061`, which is the decimal number 97. But uppercase "T" has a hexadecimal value of `\u0054`, which is the decimal number 84. 84 is lower than 97, so the "T" comes first.

For manipulating strings of text we could use some strings methods: `toUpperCase`; `toLowerCase`; `compareTo`; `IndexOf`; `endWith`; `startsWith`; `Substring`; `Equals`; `charAt`; `trim`; `valueOf`.

Converting to Upper and Lower Case in Java

Converting your Java strings of text to upper or lowercase is fairly straightforward. Just use the inbuilt methods `toUpperCase` and `toLowerCase`.

Study the illustrating example below to gain more understanding.

```
String changeCase = “text to change”;
System.out.println (changeCase);
//create result and assign the new case to it.
String result;
result= changeCase.toUpperCase();
System.out.println (result);
```

The first two lines of code just set up a String variable to hold the text “text to change,” and then we print it out. The fifth line is where we do the converting.

```
result = changeCase.toUpperCase();
```

Thereafter, the result is printed out.

Compare Strings

You can compare one string to another. When comparing, Java will use the hexadecimal values rather than the letter themselves. For example, if you wanted to compare the word “Ape” with the word “App” to see

which should come first, you can use an inbuilt string method called "compareTo."

For instance:

```
int result;
String Word1= "Ape";
String Word2= "App";
result = Word1.compareTo (Word2);
if (result < 0) {
      System.out.println ("Word1 is less than
Word2");
}
else if (result >0) {
      System.out.println ("Word1 is greater
than Word2");
}
else {
      System.out.println ("Word1 and Word2 are
the same");
}
```

The compareTo method returns a value. The value that is returned will be greater than 0, less than 0 or have a value of zero (0). If Word1 comes before Word2, then the value that is returned will be less than 0. If Word1 comes after Word2 then the value returned will be greater than

0. If the two words are identical then a value of 0 will be returned.

In essence, when you compare one string of text with another, Java compares the underlying hexadecimal values, rather than the actual letters. Because uppercase letters have a lower hexadecimal value than lowercase ones, an uppercase letter "A" in "App" will come before a lowercase letter "a" in "ape" in your code.

Notwithstanding, the `compareToIgnoreCase` method could be used to compare lowercase and uppercase letters ignoring their case.

The Java Method; IndexOf

The `indexOf` method is used to locate a character or string within another string. For example, you can use it to see if there is a @ character in an email address:

```
char ampersand = '@';
String email_address = "move@me.com";
```

```
int result;
result = email_address.indexOf (ampersand);
System.out.println (result);
```

We want to check if the @ sign is in the email address, so we first set up a char variable and assign it a value of '@.' Remember that single quotes are for char variable. After setting up an email address, we have a result variable. This is an `int` variable. The reason that result is an integer is because the `indexOf` method will return a value. It will store the value in the result variable. The `indexOf` method counts starting with 0.

However, if the character is not in the word that you're searching, `indexOf` will return a value of -1. The returned value is of great advantage. You can enjoy this advantage using the `if` statement.

Ends With… Starts With

For the illustration above, you can also use the inbuilt method "`endsWith`"; for instance:

```
Boolean ending = email_address.endsWith(dotcom);
```

You need to set up a Boolean variable for `endsWith`, because the method returns an answer of true or false. The string you're trying to test goes between the round brackets of `endsWith`, and the text you're searching goes before it. If the text is in the search string then a value of true is returned, else it will be false. You can add an `if…else` statement to check the value:

```
if (ending == false) {
      System.out.println ("Invalid Email
Address");
}
else {
      System.out.println ("Email Address OK");
}
```

The method `startsWith` is used in similar way:

```
 Boolean startsVal= email_address.startsWith
(dotcom);
```

Substring

`Substring` is a method that allows you to grab one chuck of text from another. For example:

```
String fullName= “Noble Noel”;
String surname= fullName.substring (0, 5);
System.out.println (surname);
```

The equals Method

The `equals` method is used to check two strings to see if they are the same. For instance:

```
1.    String email_address1 = “meme@me.cob”;
2.    String email_address2 = “meme@me.com”;
3.    Boolean isMatch = false;
4.    isMatch = email_address1.equals (email_address2);
5.    if (isMatch == true) {
6.          System.out.println (“Email addresses match”);
7.    }
8.    else {
9.          System.out.println (“Email addresses don’t matc
10.   }
```

In the above code, we want to check if one email address is the same as another. The first two lines set up

two string variables, one for each email address. The third line sets up a Boolean variable. That's because the equals method returns a value of true or false. The fourth line is where we use the method:

```
isMatch=email_address1.equals(email_address2);
```

In between the round brackets of the `equals` method, you place the string you're trying to check. The other string goes before the `equals` method. Java will then tell you (true or false) whether the two are the same. The `if` statement checks which one it is.

The `equals` method only compares objects, however, it's OK for strings, because they are objects. But you can't use the `equals` method to compare `int` variables. For example, this code would get you an error:

```
int num1= 12;
int num2= 13;
Boolean isMatch= false;
isMatch= num1.equals (num2);
```

The `int` variable is a primitive data type, and not an object. You can turn the primitive `int` data type into an object, though:

```
int num1= 12;
Integer num1 = new Integer (num1);
```

Here, the `int` variable called num1 is being turned into an integer object.

The charAt Method

The `charAt` method is used in Java to check to see which single character is in a particular string. For example:

```
String email_address= "meme@me.com";
char aChar= email_address.charAt (4);
System.out.println (aChar);
```

This code checks which letter is at position 4 in the email address string. The return value is a variable of type `char`:

```
char aChar= email_address.charAt (4);
```

When the code is run, the output is the character @. The number between the round brackets of `charAt` is the position in the string you're trying to check. Here, we want to get at the character in position 4 of the email_address string. The count starts at 0, just like substring.

One good use for `charAt` is for taking a letter from a string variable that is typed by a user, and then converting it to a single char variable. For example, you could ask the user to type `Y` to continue or an `N` to exit.

```
Scanner user_input= new Scanner (System.in);
System.out.println ("Quit? Y/N");
String aString= user_input.next();
char aChar = aString.charAt(0);
if (aChar == 'Y') {
      System.out.println ("OK, BYE BYE");
}
else {
      System.out.println ("Not Quitting");
}
```

We can't use the scanner class directly to get a single letter to store in a char variable. So we use the `next ()` method to get the next string that the user inputs.

There's a next integer, next long, next double–even a next Boolean; but there is no next char. Even if the user inputs a single character it will still be a string and not a char.

We can use `charAt` to get a character from any string that the user inputs, even if the user inputs a single letter.

```
char aChar = aString.charAt(0);
```

The if statement checks the typed letter and then prints its appropriate information.

The replace method

The `replace` method in Java is used to replace all occurrence of a character(s) in a particular string. For example: in the string "Where are you books?" we could replace "you" with "your."

```
String aString = "Where are you books?";
String amend = aString.replace ("you", "your");
System.out.println (amend);
```

The trim method

You can trim whitespaces from your strings. Whitespace is things like space characters, tabs and newline characters, i.e. the characters you can't see. For instance:

```
String amend= "white space";
amend = amend.trim ();
```

So the trim method goes after the string you want to amend. The blank characters before after the phrase is deleted.