7 Characters & Strings

1 Review of characters

Characters includes the letters of the alphabet in both upper and lower case, the decimal digits, punctuation characters, other common and not so common symbols (such as @ or #), and various control characters (such as the new line character – written \n – which when printed causes the cursor to move to the start of a new line). The character type in Java is written `char`.

Character values are bracketed by single quote marks, as in

```
char ch =  'X';.
```

The following are some of the more useful operations on characters. They are all boolean-valued static functions in a class called Character in the Java library. Each takes a character argument (called ch below) and yields a boolean value as indicated by the name (e.g. Character.isDigit(ch) yields true if ch is a digit).

Character.isDigit(ch)
Character.isLetter(ch)
Character.isLetterOrDigit(ch)
Character.isLowerCase(ch)
Character.isUpperCase(ch)
Character.isWhitespace(ch)

For example, Character.isLetter('b') yields true, whereas Character.isLetter('1') yields false. Note that Character.isDigit(1) is incorrect – you must write Character.isDigit('1'). The following static functions are character-valued:

Character.toLowerCase(ch)
Character.toUpperCase(ch)

For example, Character.toLowerCase('B') yields 'b', Character.toLowerCase('b') yields 'b', and Character.toLowerCase('2') yields '2'.

Note that for ch a character variable, Character.toLowerCase(ch) yields a character result without changing the contents of ch. If ch is the name of a character variable and you really want to replace the contents of ch with a lower case version, you write an assignment statement: `ch = Character.toLowerCase(ch);`.

The following piece of code shows how to read a single character from the keyboard and determine if it is a letter or not:

```
char ch = Console.readChar();
if (Character.isLetter(ch))
        System.out.print("That's a letter!");
else
        System.out.print("That's not a letter!");
```

If you need further understanding of the behaviour of the character operations, look them up in a textbook.

2 Review of strings

Java provides the type String, whose values consist of all strings. A string is a sequence of characters, whether letters, digits, punctuation marks, etc. In Java, strings are delimited by double-quote marks, as in "This is a string", but the quote marks are not part of the string and do not get printed.

Strings occur most commonly as arguments of print() and println(). However, strings are values in their own right, and can be the subject of operations. The most common operation on strings is concatenation, denoted by +. For example, execution of

```
String s = "Fee Fi";
s = s + "Fo Fum";
System.out.print(s);
```

causes `Fee FiFo Fum` to be displayed. The concatenation operation also supports concatenation of single characters; for example, "cat" + 's' yields "cats" (as does "cat" + "s").

It is important that you understand the distinction between writing variable names surrounded by quotes, on the one hand, and without quotes on the other hand. The following program illustrates this; it outputs `j = 17`

```
class What {
   public static void main(String[] args) {
      int j = 17;
      System.out.print("j = " + j);
   }
}
```

-- in `"j = " + j` the first j appears on the screen literally as the letter j, while the second j appears on the screen as the contents of the variable j, i.e. 17, and so j = 17 appears at the end.

Java supplies a large collection of string operations. Some of the more important ones are listed below; note that they are all functions. Many of the operations rely on the fact that each character in a string has a position, the position of the first character in the string being 0. For example, the position or index of 'D' in "Dublin" is 0, the index of 'u' is 1, and so on. In the following table of examples, we presume that variable s is initialised as follows:

```
String s = "Dublin";
```

Expression	*Value*	*Explanation*
s.length()	6	number of characters in string s
s.startsWith("Dub");	true	does s begin with "Dub"?
s.startsWith("dub");	false	does s begin with "dub"?
s.endsWith("in");	true	does s end with "in"?
s.indexOf("bli");	2	index of first "bli" in s?
s.indexOf("cat");	-1	index of first "cat" in s (-1 as no occurrence)
s.substring(2,5);	“bli”	that part of s indexed from 2 to (but excluding) 5
s.substring(2);	“blin”	the tail of s starting from index 2
s.charAt(2);	'b'	the character at index 2 in s
s.toUpperCase();	“DUBLIN”	the upper case equivalent of s
s.toLowerCase();	“dublin”	the lower case equivalent of s
“ hi there ”.trim()	“hi there”	“ hi there ” without leading or trailing blanks

The string preceding the dot in these operations can be any string expression, and not necessarily a string variable. For example, `(s.substring(2,5)).trim()` is a valid string expression. Note that these operations do not change the string – they generate a new string which can be printed or assigned to a variable. For example, if you want to change all the lower case letters in a string s to uppercase, write the assignment

```
s = s.toUpperCase();
```

Example

The following program illustrates some string operations. It reads a person's name in the form of one or more forenames followed by a surname, possibly with spaces surrounding each word. It displays the name as upper case surname followed by the initial letter of the first forename. An example of input/output is

```
Enter name:    Wolfgang      Amadeus  Mozart
MOZART, W.
```

The program is

```
class Names {

      public static void main(String[] args) {
         // Read name and remove leading and trailing spaces
         System.out.print("Enter name: ");
         String name = Console.readString();
         name = name.trim();
         // Locate first letter of surname
         int i = name.length()-1; // at final letter
         while (name.charAt(i) != ' ') {
            i--;
         }
         // Extract the surname (and convert to upper case), & initial
         String surname = (name.substring(i+1)).toUpperCase();
         char initial = name.charAt(0);
         System.out.println(surname + ", " + initial +  ".");
   }
}
```

As an exercise, replace the two assignments to `name` with a single assignment which both reads the name and removes the leading and trailing spaces.

3 Comparing strings

Never use the relational operator == to compare strings for equalityu. Instead of == use the Boolean-valued function equals() (or equalsIgnoreCase() when the case of the characters in the arguments is not significant). For example, to compare string variables s and t for equality write one of

```
s.equals(t)
t.equals(s)
```

-- do not write `s==t` or `t==s`.

Strings can be compared according to their dictionary order, but you may not use the relational operators (<, <=, et.) for this. Instead use the integer-valued function compareTo(), as in

```
s.compareTo(t)
```

which yields a negative integer if s comes before t in the usual lexicographic (i.e. dictionary or alphabetic) ordering, a positive integer if t comes before s, and zero if s and t are equal. If you are not familiar with this, look it up in your textbook.

Banana Skin: equals() and compareTo() are used to compare strings, but characters are compared using ==, < , <= etc.

Example

The following program illustrates string comparisons. It reads two names (forename & surname, surrounded by arbitrary blanks), and displays that name which comes first in phonebook ordering. Phonebook ordering is alphabetically by surnames, with ties being resolved by alphabetic ordering of forenames. An example of i/o is:

```
Enter first name: Albert Einstein
Enter second name: Stephen Hawking
Albert Einstein
```

```
class LeastName {

    public static void main(String[] args) {
       // Read first name
       System.out.print("Enter first name: ");
       String forename1 = Console.readToken();
```

```
            String surname1 = Console.readToken();
            // Read second name
            System.out.print("Enter second name: ");
            String forename2 = Console.readToken();
            String surname2 = Console.readToken();
            // Compare names alphabetically
            if (surname1.compareTo(surname2)<0 ||
               (surname1.equals(surname2) && forename1.compareTo(forename2)<0))
                  System.out.println(forename1 + " " + surname1);
            else  System.out.println(forename2 + " " + surname2);
        }
}
```