**CS211. Spring 2004. Recitation 1**.

Topics to be covered:

1. newsgroups
2. Types char and String.
3. If there is time (there may not be time), the difference between String and StringBuffer

**1. Newsgroups**: Left to the instructor.

**2. Type char.**

(1) Note that "A" is of type String, while 'A' is of primitive type char

(2) Literals of type char are written with single quotes: 'A', '3', '#', etc. Here are possible escape sequences:

| | |
|---|---|
| '\\' backslash \ | '\'' single quote ' |
| '\"' double quote " | '\n' new line |
| '\r' carriage return | '\t' tab |
| '\f' form feed | '\b' backspace |

(3) The old ASCII representation of characters used 1 byte per character. Java uses instead the new Unicode representation of chars: 16 bits. It allows just about all the languages of the world to be encoded. Here's the representation of standard chars, which turns out to be the same as for ASCII.

They are given in hexadecimal. Hexidecimal uses the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Thus, decimal 15 is written as F and decimal 16 as 10. This means that 'A' is hexadecimal 41 (decimal 65), and 'b' is hexadecimal 61 (decimal 97).

(4) UNICODE FOR THE STANDARD CHARACTERS

| Java | Char | Java | Char | Java | Char |
|---|---|---|---|---|---|
| \u0020 | | \u0040 | @ | \u0060 | ` |
| \u0021 | ! | \u0041 | A | \u0061 | a |
| \u0022 | " | \u0042 | B | \u0062 | b |
| \u0023 | # | \u0043 | C | \u0063 | c |
| \u0024 | $ | \u0044 | D | \u0064 | d |
| \u0025 | % | \u0045 | E | \u0065 | e |
| \u0026 | ; | \u0046 | F | \u0066 | f |
| \u0027 | ' | \u0047 | G | \u0067 | g |
| \u0028 | ( | \u0048 | H | \u0068 | h |
| \u0029 | ) | \u0049 | I | \u0069 | i |
| \u002A | * | \u004A | J | \u006A | j |
| \u002B | + | \u004B | K | \u006B | k |
| \u002C | , | \u004C | L | \u006C | l |
| \u002D | - | \u004D | M | \u006D | m |
| \u002E | . | \u004E | N | \u006E | n |
| \u002F | / | \u004F | O | \u006F | o |
| \u0030 | 0 | \u0050 | P | \u0070 | p |
| \u0031 | 1 | \u0051 | Q | \u0071 | q |
| \u0032 | 2 | \u0052 | R | \u0072 | r |
| \u0033 | 3 | \u0053 | S | \u0073 | s |
| \u0034 | 4 | \u0054 | T | \u0074 | t |
| \u0035 | 5 | \u0055 | U | \u0075 | u |
| \u0036 | 6 | \u0056 | V | \u0076 | v |
| \u0037 | 7 | \u0057 | W | \u0077 | w |
| \u0038 | 8 | \u0058 | X | \u0078 | x |
| \u0039 | 9 | \u0059 | Y | \u0079 | y |
| \u003A | : | \u005A | Z | \u007A | z |
| \u003B | ; | \u005B | [ | \u007B | { |

| | | | | | |
|---|---|---|---|---|---|
| \u003C | < | \u005C | \ | \u007C | \| |
| \u003D | = | \u005D | ] | \u007D | } |
| \u003E | > | \u005E | ^ | \u007E | ~ |
| \u003F | ? | \u005F | _ | | |

(5) Type **char** is an integral type. If you execute

```
int i= 'A';
```

i will contain the integer representation of 'A', i.e. the decimal integer 65. You can also go the other way:

```
char c= 65;
```

System.out.println(c) will result in: A

You can put in explicit casts, if you wish, but they are not needed:

```
i= (int) 'A';
char c= (char) 65;
```

Below is a loop that prints AaBbCc ... Zz. Important things to note.

1. comparison of characters: c <= 'Z'.
2. character arithmetic: c++, 'a' - 'A'
3. The arithmetic 'a' - 'A' and c+diff actually converts the chars to integer and then does the arithmetic operation. The cast
   (char) (c+diff)
   is necessary; otherwise, the integer representation of the char, and not the char, will be printed.
4. Note the use of the characters themselves, 'a' and 'A', in computing the difference between them. It is TERRIBLE programming practice to use their representations (e.g. 97-65). DON'T DO THAT!

```
// Print AaBbCc ... Zz
int diff= 'a' - 'A';
// invariant: Capital letter c is the next letter to be printed
for (char c= 'A'; c <= 'Z'; c++) {
    System.out.print(c);
    System.out.print( (char)(c+diff) );
}
```

**3. Types String and StringBuffer**

Some students don't know much about classes yet; take this into account when discussing Strings.

About notation. We use h..k to denote the sequence of integers h, h+1, h+2, …, k. Thus, h..h+1 denotes the integers h and h+1; h..h denotes the single integer h, and h..h-1 denotes the empty range of integers. The latter is important, mathematicaly speaking, and we make heavy use of it. When writing h..k, it is assumed that h+1 ≤ k. If h is much bigger than k, it is an error.

The number of values in the range h..k is: k+1-h. Remember that.

We use the notation to help us describe substrings of Strings and segments of an array. For example, if s is a String, s[5..7] is the substring consisting of characters s[5], s[6], and s[7]. If b is an array, that b[0..5] is the subarray consisting of the first 6 elements. Also, b[0..-1] is the empty array segment beginning at b[0].

1. s= "abc"; stores in String variable s the String "abc". String literals are written with double quotes. 'A' is of primitive type char. "A" is a String that contains a single character, 'A'.

2. "" is the empty string.

3. "...".length() is the number of characters in String "...".

4. You can use the escape sequences in Strings. Consider the following:

```
System.out.println("ab\ncd\n345\".");
```

This prints:

```
 ab
cd
345".
```

Remember that System.out.println(arg) prints its argument on the Java console and then does a "new line". There's also System.out.print(arg).

5. Catenation. If either s1 or s2 is a String, then s1 + s2 denotes String catenation. "ab" + "cd" is the String "abcd". If one operand is not a String, it will be converted to a String. Make a point about the last one!!!

```
System.out.println(2+"bc");       prints:  2bc
System.out.println(2 + 3 + "bc"); prints:  5bc
System.out.println((2+3) + "bc"); prints:  5bc
System.out.println("bc" + (2+3)); prints:  bc5
System.out.println("bc" + 2 + 3); prints:  bc23
```

6. Suppose a method p requires a String as a parameter and you want to give it char c as a parameter. Use the following:

```
char c= 'A';
...
 p("" + c);
```

Just p(c) would yield a type error --the argument is not a String. But with argument "" + c, the char is first converted to String and then catenated to the empty String.

7. Checking for equality of Strings s and t. USE s.equals(t) and NOT s == t. We explain it more when we get to classes.

8. Useful methods on a String s. Remember, chars are numbered 0, 1, 2, ... s.length()-1.

| | |
|---|---|
| s.charAt(i): | the char at position i. |
| s.substring(i). | the substring s[i..s.length()-1]. |
| s.substring(i,j). | the substring s[i..j-1]. |
| s.trim() | s with whitespace removed from front and rear |

9. Class StringBuffer. Objects of class String are *immutable*. You can't change them. Objects of class StringBuffer are *mutable*; You can change them. Look at this:

```
StringBuffer sb= new StringBuffer("xyz");
sb.append("w");                    // After this, sb contains "xyzw"
sb.deleteCharAt(1);                //  After this, sb contains "xzw"
sb.reverse();                      // After this, sb contains "wzx"
```

StringBuffer has the usual methods charAt, length, substring, and toString. It also has methods for deleting a substring, insert a string at a position, replacing a substring by another string, and changing the length of the StringBuffer.

Here is how one usually proceeds. Use class String most of the time. Whenever a large amount of processing has to be done on a String s, where the processing requires changing it in some way:

(1) StringBuffer sb= new StringBuffer(s); // Create StringBuffer sb that contains s,
(2) Process sb; // This processing can change sb, without having to make a copy of it.
(3) s= sb.toString(); // Store in s a string that is the same as the string in sb.