1 Review of Basic Java

1 Background

The course is a follow-on to your first programming course in Java. It is assumed you are already familiar with the basic elements of Java (such as Console, print, int, double, boolean, char, String, variables and assignment, arrays, if-while- and for-statements, static methods, parameters). We will do some revision, however, in particular methods and parameters.

2 Revision programs

Below are some elementary programs to refresh your understanding. You should be able to understand them fully.

Example: Larger of two integers
The following program reads two integers and calculates the larger of the two. An example of input/output is (input is shown in italics):

```
Enter two integers:
-17
23
23 is the larger.
```

```
class LargerOfTwo {

        public static void main(String[] args) {
                int m, n, larger;
                System.out.println("Enter two integers: ");
                m = Console.readInt();  n = Console.readInt();
                if (m>n) {
                        larger = m;
                }
                else {
                        larger = n;
                }
                System.out.println(larger + " is the larger.");
        }

    }
```

Note that when chain brackets enclose just a single statement they can be omitted. For example, the if-statement in the preceding program could be equally well written as

```
    if (m>n) larger = m;
    else larger = n;
```

Example: Counting letters and digits
The following program reads a line of text and counts the number of letters and the number of digits. A typical execution looks like (input is shown in italics):

```
Enter a line of text...
This is 2001.
Number of letters 6
Number of digits 4
```

The program uses method readChar() from class Console; it reads a single character from the keyboard.

```
class CharCount {
        public static void main (String argv[]) {
                int numLetters = 0; int numDigits = 0;
                System.out.println("Enter a line of text...");
                char c = Console.readChar();
                while (c != '\n') {
                        if (c>='0'&& c<='9')  numDigits++;
                        else if ((c>='A' && c<='Z') || (c>='a' && c<='z'))
                                                numLetters++;
                        c = Console.readChar();
                }
                System.out.println("Number of letters: " + numLetters);
                System.out.println("Number of digits: " + numDigits);
        }
```

Note that the end-of-line character is denoted by `'\n'`. When you wish to read an input until a certain value is entered (in this example, an end-of-line character), you must use a while-loop rather than a for-loop, and you must input both before the loop and at the end of the loop body (seek out for the two instances of `c = Console.readChar()` above). You should understand how &&'s and ||'s are used to encode complex conditions: above, `c>='A' && c<='Z'` encodes "c is an upper case letter", and `(c>='A'&& c<='Z') || (c>='a'&& c<='z')` encodes "c is either an upper case or a lower case letter". You should also understand that if no boolean condition in an if-statement evaluates to True, then the if-statement has no effect. For example, if the value read into c is a punctuation mark (such as a comma or period), then no incrementing of numLetters or numDigits takes place.

Example: Best student

In the following larger example, the program reads a 100 lines from the keyboard. Each line contains a student mark followed by the name of the student. The program displays the name of the best student (i.e. the one with the highest mark). An example of input is:

```
57 Michael Murphy
89 Patrick James McMahon
78 Jenny Smith
.....
```

This will give rise to an output such as

```
The best student is Patrick James McMahon who scored 89 marks.
```

```
class BestStudent{
        public static void main(String[] args) {
                final int numStudents = 100; // number of students
                int bestMark = -1; // Best mark seen so far (-1 for clean start)
                String bestStudent = ""; // Name of best student so far.
                int i = 0; // number of students read so far
                while (i<numStudents) {
                        int m = Console.readInt();  String s = Console.readString();
                        // Is this student the best so far?
                        if (m>bestMark) {  // Yes, so note the details
                                bestMark = m;  bestStudent = s;
                        }
                i++;
            }
            // Print result
            System.out.println("The best student is " + bestStudent +
                                        " who scored " + bestMark + " marks.");
        }
    }
```

3 Type conversion reviewed

The number 2000, say, can be represented in a program in several ways. If 2000 arises as, say, the maximum number of data items that the program is prepared to process, then 2000 is best represented as an integer. On the other hand, if 2000 arises as the average of a collection of real numbers and it is just accidental that it is a whole number, then we think of 2000 as being the real number 2000.0. Finally, if 2000 is a telephone number then it is appropriate to represent it as the string “2000”. (It would be quite wrong to treat a telephone number as an integer, because many telephone numbers start with 0, and the integer type does not distinguish between, say, 08612345 and 8612345.)

Similar remarks hold for other values. For example, a letter of the alphabet can be represented as a character or as a string of length 1. Usually it is appropriate to represent it as a character of course, but not always.

Occasionally we need to convert representations, i.e. we need to translate a value represented in some type to a corresponding representation in another. For example, we may need to convert the string “2000” to the integer 2000. This is called “type conversion”. The usual mechanism in Java for type conversion is called “type casting”: we write (T) x to convert x to type T. We give some examples of the use of this below, and discuss some other useful type conversion mechanisms.

Converting reals to integers

You can convert a real to an integer in either of two ways. The first way is to prefix the real with (int), as in either of the following

```
(int) 3.14
(int)(3.14*2.1).
```

The second pair of brackets in (int)(3.14*2.1) indicates that the type conversion is to apply to the entire expression and not just the first term. The fraction is lost in the conversion. For example, (int) 3.95 is 3.

It is usually more appropriate to round off a real before converting it to an integer. For x any real, Math.round(x) yields x rounded off; for example, Math.round(3.95) yields 4.0, and Math.round(3.14) yields 3.0. The best way to convert a real x to an integer is (int) Math.round(x). For example,

```
(int)Math.round(3.95)  =  4.
```

Converting integers to reals

To convert an integer to a real, prefix it with (double), as in

```
(double) 4
```

which yields 4.0. Actually, you don’t often have to so this because Java allows you to supply an integer wherever a real would normally be expected; the conversion takes place automatically. For example, it is legal to write 3.14*27 (the result will be a real).

Converting strings to integers

If a string s encodes an integer (such as "37" or "-37", say, but not any of "3.14" or "37.0" or "1+3", or "three") then Integer.parseInt(s) yields the encoded value as an int type. For example,

```
Integer.parseInt("12") =  12.
```

Converting integers and reals to strings

In many cases, you can supply a number where a string would normally be expected, and java will convert it to the corresponding string automatically. You can exploit this to convert a number to a string: just concatenate it with the empty string. For example,

```
""+37  =  "37"
```

(note that the + here denotes string concatenation, not addition). This trick works because Java recognises the first argument as a string, treats the + as concatenation, and automatically converts the second argument to a string. Here is one more example:

```
String year = "2001"; // any year
String nextYear = "" + (Integer.parseInt(year)+1); // computes following year
```

The outer brackets in (Integer.parseInt(year)+1) are necessary in order for the inner + to be interpreted as integer addition rather than string concatenation.

Reals are converted to strings similarly; e.g.

""+3.14 = "3.14".

Converting strings to reals

If a string s encodes a real number (such as "37.3" or "-37.3") then Double.parseDouble(s) yields the encoded value as a double. For example,

Double.parseDouble("3.14") = 3.14.

Summary

x	**int**	**double**	**char**	**String**
int		`(double) x`	`(char)x`	`""+x`

double	`(int) x`
	`Math.round(x)`
				`""+x`

char	`(int) x`			`""+x`
String	`Integer.parseInt(x)`	`Double.parseDouble(x)`		