

Formatting, Input, Wrapper Classes, Special Operators

I. Formatting Output using NumberFormat

Review: given: **double** x =10.5; and we want the output to be \$10.50 we could use the format technique we learned in the last unit that works with only the Ready IDE:

```
c.print("$");  
c.println(10.5, 5, 2);
```

A) Formatting Currency (money) using “real” Java

We can have the computer automatically take a number like 10.5 that represents \$10.50 and output it as \$10.50 and not 10.5 by using an object from the class NumberFormat which is in the package java.text .

Note: **import** java.text.NumberFormat would need to be included at the top of the Java program

To use the NumberFormat class and its built-in methods you first need to create an object by declaring it:

```
NumberFormat z;
```

then you must actually create it:

```
z = NumberFormat.getCurrencyInstance( );
```

Note: `getCurrencyInstance()` is a supplied (built-in) method that technically returns an object of the appropriate type for `z`

Note the above two statements can be combined into one line:

```
NumberFormat z = NumberFormat.getCurrencyInstance( );
```

To output the newly formatted `z` object you then use the `format()` method which is also a supplied (built-in) method of the NumberFormat class:

```
c.print(z.format(10.5));
```

Now lets put it all together in a program:

```
import java.text.NumberFormat;
import hsa.Console;
public class P1U3Ex1
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        double x = 10.5;
        NumberFormat z = NumberFormat.getCurrencyInstance( );
        c.print(z.format(x));
    }
}
```

Output:
\$10.50

B) Formatting Percents

We can have the computer automatically take a number like .05 that represents 5% and output it as 5% and not 0.05 by using an object from the class NumberFormat

To use the NumberFormat class and its built-in methods you first need to create an object by declaring it:

```
NumberFormat r;
```

then you must actually create it:

```
r = NumberFormat.getPercentInstance( );
```

Note: getmyPercentExInstance() is a supplied (built-in) method that technically returns an object of the appropriate type for r

Note the above two statements can be combined into one line:

```
NumberFormat r = NumberFormat.getPercentInstance( );
```

To output the newly formatted r object you then use the format() method which is also a supplied (built-in) method of the NumberFormat class:

```
c.print(r.format(10.5));
```

Now lets put it all together in a program:

```
import java.text.NumberFormat;
import hsa.Console;
public class P1U3Ex2
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        double y = .05;
        NumberFormat r = NumberFormat.getPercentInstance( );
        c.print(r.format(y));
    }
}
```

Output:

5%

ex. What if y above was 10.5, what would be the output?

Answer: 1050%

Optional: (not discussed in our book)

By changing:

```
NumberFormat myPercentEx = NumberFormat.getCurrencyInstance( ); to
NumberFormat myPercentEx = NumberFormat.getCurrencyInstance(Locale UK);
```

the output will be in English pounds and other countries can be inserted like FRANCE or ITALY etc.

C) Formatting Numbers - Method 1 (Note: not covered in our book)

We can have the computer automatically take a number like 1234.56789 and output it with commas and as many decimal places as desired like 1,234.57 or take a number like 1234 and output it with significant zeros after the decimal point like 1,234.000 by using an object from the class NumberFormat

Note: **import** java.text.DecimalFormat would need to be included at the top of the Java program

To use the NumberFormat class and its built-in methods you first need to create an object by declaring it:

```
NumberFormat myNumberEx;
```

then you must actually create it:

```
myNumberEx = NumberFormat.getNumberInstance( );
```

Note: getNumberInstance() is a supplied (built-in) method that technically returns an object of the appropriate type for myNumberEx

Note the above two statements can be combined into one line:

```
NumberFormat myNumberEx = NumberFormat.getNumberInstance( );
```

Before outputting you must also call two other methods of the NumberFormat class:

```
myNumberEx.setMaximumFractionDigits(2);  
myNumberEx.setMinimumFractionDigits(2);
```

Note: The above two set the number of decimals to be displayed. In the above the number will be **rounded** to 2 decimal places.

To output the newly formatted myNumberEx object you then use the format() method which is also a supplied (built-in) method of the NumberFormat class:

```
c.print(myNumberEx.format(1234.56789));
```

Computer Programming I - Unit 3 Lecture

5 of 12

Now lets put it all together in a program:

```
import java.text.NumberFormat;
import hsa.Console;
public class P1U3Ex3
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        double number = 1234.56789;
        NumberFormat myNumberEx = NumberFormat.getNumberInstance( );
        myNumberEx.setMaximumFractionDigits(2);
        myNumberEx.setMinimumFractionDigits(2);
        c.print(myNumberEx.format(number));
    }
}
```

Output:

1,234.57

```
import java.text.NumberFormat;
import hsa.Console;
public class P1U3Ex4
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        double number = 1234;
        NumberFormat myNumberEx = NumberFormat.getNumberInstance( );
        myNumberEx.setMaximumFractionDigits(5);
        myNumberEx.setMinimumFractionDigits(3);
        c.print(myNumberEx.format(number));
    }
}
```

Output:

1,234.000

C) Formatting Numbers - Method 2

We can have the computer automatically take a number like 1234.56789 and output it (without commas) with as many decimal places as desired like 1234.57 or take a number like 1234 and output it with significant zeros after the decimal point like 1234.000 by using an object created of type class **DecimalFormat**

To use the DecimalFormat class and its built-in methods you first need to create an object by declaring it:

```
DecimalFormat myNumberEx2;
```

then you must actually create it:

```
myNumberEx2 = new DecimalFormat("0.0#");
```

Note the above two statements can be combined into one line:

```
DecimalFormat myNumberEx2 = new DecimalFormat("0.0#");
```

Note: This is the normal way of instantiating an object unlike the ways we did it with the NumberFormat class;

Note: The above will set the number of decimals to be displayed; one to the left of the decimal point if there is no number in that spot and two to the right of the decimal point - if there are no numbers to the right then one zero will be displayed. In the above the number will be **rounded** to 2 decimal places.

To output the newly formatted myNumberEx object you then use the format() method which is also a supplied (built-in) method of the DecimalFormat class:

```
c.print(myNumberEx2.format(1234.56789));
```

Now let's put it all together in a program:

Computer Programming I - Unit 3 Lecture

7 of 12

```
import java.text.DecimalFormat;
import hsa.Console;
public class P1U3Ex4
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        double a = 1234.56789;
        double b = 1234;
        double c = .56789;
        double d = 1234.5;
        double e = 0.8;
        DecimalFormat myNumberEx2 = new DecimalFormat("0.0#");
        c.print(myNumberEx2.format(a));
        c.print(myNumberEx2.format(b));
        c.print(myNumberEx2.format(c));
        c.print(myNumberEx2.format(d));
        c.print(myNumberEx2.format(e));
    }
}
```

Output:

```
1234.57
1234.0
0.57
1234.5
0.8
```

If the line above was: `DecimalFormat myNumberEx2 = new DecimalFormat("#.000");`
The the output would be:

Output:

```
1234.568
1234.000
.578
1234.500
0.800
```

Optional: Not included in our book is a `DateFormat` class found in the `java.text` package.

II. Input Dialog Screen

The following code shows what is needed if you want to have an interactive program with user input using an Input Dialog Screen using JOptionPane

```
import javax.swing.JOptionPane; // javax was written after java
public class ex1
{
    public static void main (String[ ] args)
    {
        String name1;
        name1 = JOptionPane.showInputDialog("Type your full name then <OK>");
        System.out.println("Your name is: " + name1);
    }
}
```

Note: the line:

```
name1 = JOptionPane.showInputDialog("Type your full name then <OK>");
```

could also be written as:

```
name1 = JOptionPane.showInputDialog(null,"Type your full name then <OK>");
(Note: the null centers the dialog window, which it usually is anyway)
```

Note: **int** num = Integer.parseInt(name1); is needed if the input is an integer

Note: **double** num = Double.parseDouble(name1); is needed if the input is an decimal number

(More about this in 3 pages)

III. Console Input - Summary for Reading from System.in

The system console uses the System.in object to obtain user input. The System.in object provides methods in its interface to read a set of bytes from the keyboard. The programmer, however, cannot use these methods directly (unlike System.out).

Before the system console can use the data from the keyboard, the data must be translated from bytes provided by the keyboard into Unicode characters. The characters then must be formatted to allow the user to read a line of data from the keyboard. This is done by routing the data from the keyboard through two objects.

The first object is an InputStreamReader object that translates the bytes produced by the keyboard into Unicode. The second object is a BufferedReader object that translates the Unicode characters into a line of text. Here is the code to create an object that the programmer can use to read lines of data from the keyboard using the System.in object:

```
InputStreamReader instream;  
BufferedReader keyboard;  
inStream = new InputStreamReader(System.in);  
keyboard = new BufferedReader(inStream);
```

The first line declares the variable inStream to be a reference to an object of the InputStreamReader class. The second line declares the variable keyboard to be a reference to an object of the BufferedReader class. The third line creates the InputStreamReader object using the new keyword and assigns a reference to the object to inStream. The last line creates the BufferedReader object and assigns a reference to the object to variable keyboard.

To use the InputStreamReader and BufferedReader classes, the programmer must include import statements at the top of the program. These import statements inform the Java compiler where the InputStreamReader and BufferedReader classes are located. In this case, the two classes are located in a group of classes (called a package) called java.io. This package contains classes to perform input and output to and from files. The statements placed in the import section at the top of the program are:

```
import Java.io.InputStreamReader;  
import Java.io.BufferedReader;
```

Computer Programming I - Unit 3 Lecture

10 of 12

The program uses the `BufferedReader` object to read input from the system console. The `BufferedReader` class has several methods in its interface. The relevant one here is a function-type method called `readLine()`. This method takes no arguments and returns a line of input typed by the user at the system console. If there is no line of input waiting, the program waits until the user has entered a line of input and then returns a string object that contains the line that the user entered (without the Enter that the user pressed at the end of the line).

```
String LineFromUser = keyboard.readLine();
```

There is one last element you must deal with. Although it is unlikely, the `readLine()` method can fail. Java requires that you at least acknowledge the possibility of failure. This is done by adding the line

```
import java.io.IOException; in the import section of the program,  
and adding the clause
```

```
throws IOException to the declaration of public static void main(String[] args)  
to make
```

```
public static void main(String[] args) throws IOException
```

This is what will produce the built-in error message in to the Java program.

```
import java.io.InputStreamReader;
```

```
import java.io.BufferedReader;
```

```
import java.io.IOException;
```

```
public class ex1
```

```
{  
    public static void main (String[] args) throws IOException  
    {  
        InputStreamReader inStream;  
        BufferedReader keyboard;  
        inStream = new InputStreamReader(System.in);  
        keyboard = new BufferedReader(inStream);  
        String input;  
        int num;  
        System.out.println("Type in an integer then <Enter>");  
        input = keyboard.readLine( );  
        num = Integer.parseInt(input);  
        System.out.println("Your input now as an integer is: " + num);  
    }  
}
```

OR a shortcut:

```
BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
```

IV. Wrapper Classes

The line: `num = Integer.parseInt(input);`
uses a method from the class `Integer` to convert the string `input` into a useable integer.

The classes `Double`, `Integer`, and `Float` are called wrapper classes and allow numbers to act as objects which is useful if you need to use a method that works on objects such as sorting. When inputting from the keyboard (console) everything is input as a string and if it is actually a number or a character then it must be converted by one of the following

```
int ex1 = Integer.parseInt(input);  
double ex2 = Double.parseDouble(input);  
float ex3 = Float.parseFloat(input);  
char ex4 = Character.parseChar(input);
```

V. Shortcut Console Input

Our lab book author has created a console input shortcut. In the `hsa` package is a class called `Stdin`

```
import hsa.Stdin;  
public class ex  
{  
    public static void main (String[ ] args) throws IOException  
    {  
        int x;  
        System.out.println("Type in an integer then <Enter>");  
        x = Stdin.readInt( );  
        System.out.println("Your input is: " + x);  
    }  
}
```

The line `x = Stdin.readInt();` could be:

```
double input = Stdin.readDouble( );    // if a double number is to be input  
char input = Stdin.readChar( );        // if one character is to be input  
boolean input = Stdin.readBoolean( );  // if true or false is to be input  
String input = Stdin.readString( );    // if one word is to be input  
int input = Stdin.readLine( );         // if a string (multiple words) is to be input
```

VI. *Special Operators*

A) *Unary Arithmetic Operators* (ones that take just one operand):

1) simple ones: + and -

ex.1) -7

ex.2) +7 (Note: The + as a unary operator is rarely used since if there is no sign then the number is automatically positive.)

2) a) *increment* and *decrement* operators: ++ and --

ex.1a) number = 10;
 number = number + 1;

or

ex.1b) number = 10;
 number++;

or

ex.1c) number = 10;
 ++number;

ex.2a) number = 10;
 number = number - 1;

or

ex.2b) number = 10;
 number--;

or

ex.2c) number = 10;
 --number;

B) Additional Java Combined Assignment Operators: +=, -=, *=, /=

ex.1) $y -= 3$ *// is the same as $y = y - 3$*

ex.2) $y *= 3 + x$ *// is the same as $y = y * (3 + x)$*

Note 1: $y =/ 3$ *// is illegal*

Note 2: $y =+ 3$ *// is legal Why?*