Handout 2

Basic Java Constructs: Variables, Primitive Types, Expressions
Keyboard input with Scanner class.
Formatting output with *printf* and DecimalFormat class.

Problem: Write a program that determines the length and cost of a phone call, given the starting and ending time based on 24-hour clock and the per-minute rate.

The program should work as follows: the user will be enter six parameters:

- the starting hour and minute, separated by white space,
- the ending hour and minute, separated by white space,
- the per-minute rate in dollars,
- the phone number from which the call was made,
- the phone number to which the call was made.

The program would then compute and output the length of the call in minutes, and the cost of the call in dollars, displaying 2 digits after the decimal period in the format shown in the example below.

Here's a sample interaction, in which user input appears in bold.

```
This program computes the length and cost of a phone call.
Please enter the hour and minute of the start of the phone call: 12 15
Please enter the hour and minute of the end of the phone call:  13 34
Please enter the per-minute cost of the call:  0.25
Please enter the phone number of the origin: 781-891-3161
Please enter the phone number of the destination: 781-948-4357

From: 781-891-3161 To: 781-948-4357:  Duration: 79 min
Rate: $0.25        Cost: $19.75
```

Design: specify

Input:

Output:

Data:

Algorithm:

Let's implement this program, simultaneously introducing basic Java concepts.

1. Comments

- any number of lines enclosed within /* */ block , or
- starting with // to the end of line

Comments are necessary for readability – element of good programming style

Include introductory comments in every program, describing the content of the program.

2. Code to begin the program

```
public class PhoneCall {


    public static void main(String[] args)
    {

    }

}
```

definition of class PhoneCall ***is enclosed within { } that follow***

definition of the main ***method of class*** PhoneCall

Java applications all have similar code at the beginning.

First line `public class PhoneCall` declares a class.
Explanation:

- `public` - is an access control keyword[1]. means can be used by other classes. For now all classes we define are going to be public.
- `class` is a keyword that must be followed by the name of the class.
- `PhoneCall` is the name of the class – you get to choose it
- **File name must be the same as the name of the class it defines.**

Next: definition of method `main`:

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

Explanation:

- `public` - similar to above
- `static` - will have to wait for explanation
- `void` - returns no value.
- `main` - **reserved** method name; `main` method of a class is an entry-point to it – **it is executed first** when a class is executed.
- `String[] args` – a list of parameters passed to the program through a command line – we will not be using it for a while.

Main method contains all the instructions of this program.

[1] Keyword – (a.k.a. reserved word) is a word that has a special meaning in a language and cannot be used for other purposes.

3. Start a program with instructions or the description to tell the user what the program does and/or how to use it:

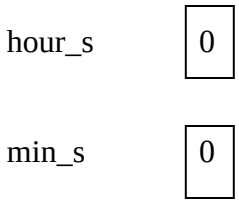
```
System.out.println("This program computes the length" +
        " and cost of a phone call.");
```

4. Variable declarations, e.g.

```
int hour_s, min_s; // hour and minute of start
int hour_e, min_e; // hour and minute of end of call
```

Variable - is a named location to store data

- It can hold only one type of data
- for example only integers, only floating point (real) numbers, or only characters
- All program variables **must** be *declared* before using them. A variable declaration associates a name with a storage location in memory and specifies **the type of data** it will store: e.g. `int, float, double, char, String`



Named constant: a variable that cannot be changed after the first assignment. Declared with keyword final, e.g.

```
final int MIN_WAGE = 7.25;
```

5. Java has Primitive and Class data types

Primitive types (e.g. `char, int, float, double, long`)

- the simplest types, they define values that cannot decompose into other types (in other words are atomic)
- Each primitive type has *operators* that apply to values of that type: e.g. numeric operations, *boolean* operators (we'll study later). Operators are used to form *expression*s.

Class types (e.g. `Scanner, DecimalFormat, String)` more complex, used to define objects

- composed of other types (primitive or class types)
- both data and methods

6. Assignment operator =

Used to **set, or assign** a variable is to store a given value in the location denoted by the variable name .

Examples:

```
1. // initialize and later decrement the variable
int count = 10;// initialize counter to ten
count = count - 1;// decrement counter

2.//whole expression on the right hand side
 int days;
 days = 366*numLeapYears + 365*(numYearsTotal-numLeapYears);
```

Not the same as equality in algebra, It means -
"Store the value of the expression on the right side to the variable on the left side."

Can have any expression on the right hand side of =
Restriction: the type of the variable **must be compatible** with the type of the expression on the right hand side.

7. Decimal numbers are represented with types **double** and **float**, e.g.

double rate = 0.33;
float limit = 2*10.567

Values of these types are only approximate representation of the numbers.

8 Characters

are actually stored as integers according to a special code: each printable character (letter, number, punctuation mark, space, and tab) is assigned a different integer code

the codes are different for upper and lower case
for example 97 may be the integer value for 'a' and 65 for 'A'

Java uses Unicode (Unicode includes all the ASCII codes plus additional ones for languages with an alphabet other than English).

Casting a char value to int produces the ASCII/Unicode value
Problem: what would the following code segment display?

```
char answer = `y';
System.out.println(answer);
System.out.println((int)answer);
```

Some special "invisible" characters: '\n' – end of line, '\t' – tab, '\''-single quote.

9. Strings

- will study in more detail later. A class type that represents sequences of characters enclosed in double quotes, e.g. `"This is a string", ""` – empty string.

10. Keyboard input using Scanner class.

Class Scanner has methods to read values entered via keyboard input.

a. Need to include the following line above the class definition

```
import java.util.Scanner;
```

This statement tells Java to
- Make the **Scanner** class available to the program
- Find the **Scanner** class in a library of classes (i.e., Java *package*) named **java.util**

b. Need to create an object of Scanner class inside the main method as follows:

```
Scanner anyName = new Scanner (System.in);
```

Once a **Scanner** object has been created, a program can then use that object to read user input from the keyboard using methods of the **Scanner** class These methods are **type-specific**

`anyName.nextInt();` - returns an integer entered by the user. Skips all white space (space, tab, end-of-line characters) that appear **before** the integer.

`anyName.nextDouble();` - double

`anyName.nextBoolean();`

`anyName.next();` - returns the String value consisting of characters up to, but not including the next white space (or end of line). Essentially, reads one word, where word is a sequence of non-white space characters.

`anyName.nextLine();` - returns the String value consisting of characters up to, but not including the end of the current line. Unlike the other methods, **does not skip the end-of-line character, instead treats it as the end of the line.**

Subtle point:

- The method **nextLine** of the class **Scanner** reads the remainder of a line of text starting wherever the last keyboard reading left off. This can cause problems when combining it with different methods for reading from the keyboard such as **nextInt**

Example: Given the code,

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

and user input,

```
2
Heads are better than
1 head.
```

what are the values of **n**, **s1**, and **s2**?

Answer: n will be set to 2, s1 will be equal to "", and s2 will be equal to "Heads are better than"

Explanation: **\n** below denotes the "invisible" end of line character. Here's what the user input looks like

```
2\nHeads are better than\n1 head.\n
```

If the following results were desired instead
n equal to 2, s1 equal to "heads are better than", and s2 equal to "1 head"
then an extra invocation of nextLine() would be needed to get rid of the end of line character '\n'.

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
keyboard.nextLine(); // to get rid of \n after number
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

This only happens when the program needs to read a new line with nextLine() after using any other Scanner reading methods (next(), nextInt(), nextDouble(), etc..)

11. Arithmetic operators.

Work on all numeric types: + - * / % ()

Order of precedence:

i. ()
ii. *, /, %
iii. +, -

Note: The result of an expression involving only of whole number types (byte, short, int, long) is always a whole number (byte, short, int, long). If at least one of the operands has real number type (double, float), the result is a real number (double, float).

Practice problems:

2.
```
int a = 5, b = 3, e;
double c = 2, d;

// what is stored in each of the assigned variables (d and e)?
d = a/b;
e = a/b;
// what is printed by the following?
System.out.println(a/b);
System.out.println(a/3.0);
```

3. What is the result of the following computations given the following declarations:
int a = 5, b = 3, c = 2 ?

```
int d = a – b * c;          int d = (a – b) * c;
int d = a – b / a - c;      int d = a / b + a % c;
```

4. Change the PhoneCall program to display the duration of the call in hours and minutes instead of just minutes.

12. Arithmetic expressions and type conversion.

Recall that the type of the variable must be compatible with the type of the expression on the right hand side. So, would the following be legal?

```
double x;
int n = 5;
x = n;
```

the value returned by n is *cast* to a double, then assigned to variable x as 5.000… (as accurately as the whole number 5 can be encoded as a floating point number).

This is called *implicit* casting because it is done automatically.

The following chart describes the allowable automatic type conversion (a.k.a. type casting):

byte → short → int → long → float → double

The logic behind it: can automatically convert so long as the new type represents a larger range of values

.

Otherwise, need to *explicitly* specify the conversion via *type casting*: e.g.

```
int n;
double x = 2.89;
n = (int)x;//legal in java. n will be set to 2.
```

NOTE: using typecasting from double to int for rounding will produce incorrect results. Use **Math.round();** instead. Math.round(3.4) → 3, Math.round (3.5)→ 4

13. Output and formatting.

- we'll use 2 methods available Java using printf and DecimalFormatter.

- **System.out.printf()** – is a method that takes any number of parameters. The first parameter must be the formatting string. Formatting string may contain placeholders for the values of other parameters that will be substituted during printing. These placeholders can define how exactly these parameters are printed.

  Placeholders (a.k.a. conversion characters, format specifiers)

  – complete description on page 64

  %d - ordinary integer

  %f - floating point

  %s - string

  %c - character.

Examples:

1. The code segment

```
double price = 19.8;
System.out.printf("The price is %6.2f", price);
```

will output the line

The price is $ 19.80

The format specifier "%6.2f" indicates the maximum number of digits displayed is 6, display exactly 2 digits after the decimal point (.2)

2. The code segment

```
String name = "Anne Lee";
int age = 25;
double GPA = 3.4;
System.out.printf("%s is %d years old and has GPA %.2f",
                                        name, age, GPA);
```

will output **Anne Lee is 25 years old and has GPA 3.40**

- **DecimalFormat class**
  - The DecimalFormat class must first be imported
  - A DecimalFormat object is associated with a pattern when it is created using the **new** command
  - The object can then be used with the method **format** to create strings that satisfy the format

  Example:

```
/* Created on Mar 3, 2005 9:26:21 AM
 * Modified textbook example.
 * Demonstrates the use of DecimalFormat class for
 * formatting the decimal values for printing.
 */

import java.text.DecimalFormat; // required import statement

public class DecimalFormatDemo {
      public static void main(String[] args) {
            // create a formatting pattern
            // 0 in the pattern stands for a required digit
            // # stands for an optional

            DecimalFormat p =  new DecimalFormat("00.000");
            DecimalFormat p1 = new DecimalFormat("0.00");
            DecimalFormat p2 = new DecimalFormat("#00.000");
            DecimalFormat p3 = new DecimalFormat("000.000");

            double d = 12.3456789;
            System.out.println("Unformatted: " + d);
                                          // prints 12.3456789
            // use the pattern to format for printing
            System.out.println("Pattern 00.000");
            System.out.println(p.format(d));    // prints 12.346

            System.out.println("Pattern 0.00");
            System.out.println(p1.format(d));   // prints 12.35

            System.out.println("Pattern #00.00");
            System.out.println(p2.format(d));   // prints 12.346

            System.out.println("Pattern 000.00");
            System.out.println(p3.format(d));   // prints 012.346

            double money = 19.8;
            System.out.println("Unformatted: " + money);
            System.out.println("Pattern 0.00");
            System.out.println("$"+p1.format(money));
                                             // prints $19.80

      }
}
```

12. Rules of good programming style.

Look at the following program and try to understand what it does.

```
import java.text.DecimalFormat;
import java.util.Scanner;
public class Mystery {
public static void main(String[] args) {
int a, b, c, d;
Scanner kbrd = new Scanner(System.in);
 System.out.println("Pls enter first two params");
a = kbrd.nextInt();     b = kbrd.nextInt();
 System.out.println("Pls enter next two params");
  c = kbrd.nextInt();
d = kbrd.nextInt();
int m1 = a*60 + b;
     int m2 = c*60 + d;
int t = m2-m1;
DecimalFormat frm = new DecimalFormat("00");
//display output
System.out.println (frm.format(t) );
}}
```

To make programs ***readable***:

- Document program with comments.
- Use **named constants** where appropriate:

Named Constants, e.g.

```
// declare a CONSTANT:
final int BENTLEY_MAX_CLASS_SIZE = 35;
```

- are similar variables, but cannot be changed.
- declared with the keyword `final`

Advantages of using named constants:
Example: use MORTGAGE_INTEREST_RATE instead of 8.5
Why?

Easier to understand a program because reader can tell from the name, what's denoted by the value.
Easier to modify a program because value can be changed in one place (the definition) instead of being changed everywhere in the program.

- Use meaningful names for variables, constants, classes, etc.
- Use indentation and line spacing as shown in the examples in the text
- Always include a "prologue" (an brief explanation of the program at the beginning of the file)