Handout 7

Static Methods

Static Methods in Java = functions or **procedures** in structured programming languages.

static method

- in other structured programming languages know as **function** or **procedure**.
- is a code module that implements a functionally independent task.

- Examples - built-in **static** methods provided in the JDK, e.g.
  - `Integer.parseInt()`
  - `Double.parseDouble()`
  - `Math.round()`

- Others are written by programmer

Methods support **structured programming**:

- Top-down construction
- Easier to distribute work and reuse the code
- Easier to debug
- Easier to follow the logic
- Reduces errors

Using methods:

- **main( )** method: where execution starts.
- Other methods *called* or *invoked* from there.
- Control passes to the called method.
- Code in method is executed.
- Control returns to where the method was called from.
- Methods can call other methods.

Example:

```
public class SimpleMethod {

     public static void main(String[] args) {
          System.out.println("in main before calling printHi");
          printHi();      // CALL, a.k.a. INVOKE method printHi;
          System.out.println("in main after calling printHi");

     }

     // definition of method printHi is below:
     public static void printHi() {
          System.out.println("This method simply prints HI!");
     }
}
```

Notes on program `SimpleMethod`:

1. Both the `main()` method and the `printHi()` method are included in the `SimpleMethod` class.

2. The **order** of the `main()` and the `printHi()` methods does not matter. Either one can appear first within the `SimpleMethod` class.

3. A **static** method is a **class** method: it is associated with the class as a whole, rather than with an *instance*, or *object*, of the class.

4. **void** means that the method does not return any information to the calling method.

5. The `main()` method **must** be **public** – part of syntax for running a Java application. In all other cases, `public` means the method can be used by other classes. Public access is optional for other methods besides `main()`.

All Java methods have

- name
- zero or more input parameters and
- one or no return value (output).

Methods may also have

- Local variables and constants

Form for static methods:

```
public static Return_type methodName(parameter list) {
    method_body
}
```

Method Body

- Statements to be executed.
- Begin or precede the method with a **comment** describing the purpose of the method, parameters and returned value.

Method Name

- Should reflect its purpose
- Convention: should begin methods with a lowercase letter.

Parameters: placeholders for data values passed to the method when it is invoked. Also called **formal parameters.**

Parameter list:

- Specify variable name and data type (e.g., int, boolean, etc.)
- Cannot be actual data values

When invoke a method, provide data as **arguments (also called actual parameters)**:

- Can be variables, in which case their types have already been defined
- Can be actual data values

Actual (passed) parameters must conform to **formal (declared) parameters**:

1. Same number, same order, same types
2. The arguments can have the same name as the parameters, but they do not have to

Example with 1 parameter:

```java
/**
 * Demonstrates a simple method with one parameter and no return value
 */
public class PrintMethod {

               // Demonstrates use of the printMany method
               public static void main(String [] args) {
                  int x = 3, y = 1, result;

                  printMany(5);      //invoke printMany with argument = 5
                  printMany(x);      // invoke printMany with argument = 3
                  printMany(x + y);  // invoke printMany with argument = 4
               }

               // definition of method printMany is below:
               public static void printMany(int nTimes) {
                  /** Uses the println method to print a message
                   the specified number of times */

                  for (int i = 1; i <= nTimes; i++)
                    System.out.println("This program simply prints HI!");
                  System.out.println();
               }
}
```

Example with 2 parameters and a **return** value:

```
/*
 * PhoneBill.java – prints cost of a phone call, given rate and length
*/

import java.util.Scanner;

public class PhoneBill {


        public static void main (String[] args) {

                Scanner kb = new Scanner(System.in);

                System.out.println("Enter length of the call in minutes");
                int len = kb.nextInt();

                System.out.println("Enter per-minute rate in dollars");
                double rate =kb.nextDouble();

                /** call computeCost method to compute the total cost
                 * of phone call.      */
                double cost = computeCost(len, rate);

                System.out. printf("The cost of the phone call is $%.2f",cost );

        }


        /**
         * public static double computeCost (int minutes, double minRate)
         *            computes and returns the cost of phone call lasting
         *            len minutes at the minRate rate
         */
        public static double computeCost (int len, double minRate) {
                double costOfCall; //local variable

                costOfCall = len*minRate;

                return costOfCall; // Terminate the method

                // Control goes back to the point from which the method was called
                // Value of costOfCall is passed back
        }

}
```

Passing back a value: different examples of the use of the `return` statement

```
return;    // used in void methods -immediately returns control
            // to the line after the one invoking the method;
return x;       // also returns the value of x
return x + y;   // also returns the value of x + y
```

- return statement terminates the method execution .
- the type of value returned must match the method's return type, specified in the method header (***`Return_type`***) .

- a method may have 0 or more return statements, but it is a rule of good programming style to have a unique exit point from a method.
- a method that does not return a value has a void return type

Scope of variables and parameters

- Local variables – variables that are declared inside a method.
  in `main` – `len, rate` and `cost`
  in `computeCost – costOfCall`
- Local variables and formal parameters are **accessible only from within the method** and are not accessible from outside of the method.
- The parameters of a method create *automatic local variables* when the method is invoked. They are set to the values of arguments passed to the method in the method call.
- When the method finishes, all local variables are **destroyed** (including the formal parameters)

Calling conventions:

- By name, e.g.

```
cost = computeCost(len, rate);
```

  works only within **the same class** (where `computeCost` is defined).

- Specifying the class name as in

```
amountDue = PhoneBill.computeCost(length, 1.12);
```

  works **within another class** as well (e.g. JOptionPane.showMessageDialog() – JOptionPane is the name of the class that contains a static methods showMessageDialog ())

Method Overloading - Ability to define multiple methods with the same name but different parameter lists.

- Methods are distinguished by their **signature**:
  - **name**
  - **number of parameters**
  - **types of parameters**.

Practice Problems:

1. Write method `isEven` that is passed an integer that returns true if and only if the number is even.
2. Create the `main` method so that it uses the `isEven` method to print whether a number is even or odd.
3. Modify `PhoneBill` program to work as follows:

   - Ask the the user to enter the per minute rate for phone call
   - open a file called `phoneCallTimes.txt,` in which each line contains an account number and an integer number, as shown

```
dg347824  32
aa45249  12
dg3wer24  25
KJds475 17
jd995t36  37
```

- produce an output file `outfile.txt` that will contain lines that show the amount due for each call, as shown (here, the rate was $0.10)

  ```
  Amount due from dg347824 is $3.20
  Amount due from aa45249 is $1.20
  Amount due from dg3wer24 is $2.50
  Amount due from KJds475 is $1.70
  Amount due from jd995t36 is $3.70
  ```

  The program must define a function `processFile()` with two parameters: name of input file and the rate. This function should open the input file and process it to produce the output file as shown.

4. Define and test a static method, which, for a file `menuitems.csv` of the following format

```
appetizer,Bouillon in cup,6.7
appetizer,Queen olives,8
entrée,"Panfish, Meuniere",25
side-dish,German fried potatoes,5
entrée,Ribs of prime beef,19
dessert,Assorted cakes,15
entrée,"Scollops en caisse, Supreme",32
entrée,Irish stew,28
appetizer,"Marrow on toast, Bordelaise",18
salad,Lobster salad,14
```

will compose and return a string containing the names of all types of dishes separated with a separator.

The method should be passed the file name and a String to be used as a separator. For example, passed the name of the file shown above and separator `“***”`, the method should return `“appetizer***entrée***side-dish***dessert***salad”`

5. What is displayed when the following code segment is executed? Someone has created this very poorly written undocumented program. Do not try to deduce the purpose of this code. Simply show the output from four System.out.println () statements in the order in which it will be displayed by the program.

Show the values of participating variables, parameters and expressions for partial credit.

```
public class Example {
      public static void main(String[] args) {
             int a = 7, v = -3;
             System.out.println("a = " + a);
             String str = Example.makeString(a, '*');
             System.out.println ( "a = " + a + " str = " + str);
             str = Example.makeString(v, '*');
             System.out.println ( "a = " + a + " str = " + str);
      }

      public static String makeString(int a, char ch){
```

```
            String result = "#";
            while (a >= 0){
                  if (a % 3 == 0)
                        result = result + ch;
                  else
                        result = result + (a/3);
                  a--;
            }
            System.out.println("in makeString a = " + a);
            return result;
      }
}
```

6. Develop a **static** method `computeChecksum()` that is going to be passed a parameter of type String . The method should verify that the string consists of digits and dashes only, and in such case it must return the sum of all digits in it.

   In case the parameter string contains any character other than a digit or a dash, the method should return -1.

   Here are a few examples:

Parameter string:	Method should return:
`“12-33-1”`	`10`
`“044”`	`8`
`“044-0-7-02”`	`15`
`“17-3457-ab”`	`-1`
`“Summertime”`	`-1`

   ***Hints*:**

   1. The simplest approach to this problem is to go through all characters in the string one by one, checking if the character is a digit, dash or anything else, and computing the total sum of all digits.

   2. You can use method `Character.isDigit()`to check if a given character is a digit. Method `Character.isDigit()` must be passed a parameter of type `char` and it returns a `boolean` value. For example, if `ch` is a variable of type `char` and `ch = ‘6’`, `Character.isDigit(ch)` will return **`true`**. If `ch = ‘a’, Character.isDigit(ch)` will return **`false`**.

   3. You can use method `Integer.parseInt()` to convert a string containing a digit into an integer. Method `Integer.parseInt()`must be passed a parameter of type `String` and it returns a value of type `int.` For example, `Integer.parseInt(“3”)` returns an integer value 3. The method will generate a run-time error if the string passed as a parameter contains any non-digit characters.

7. Compose a main method that prompts the user to enter a license string and uses the method `computeChecksum` () developed in problem 5 to test the license number entered by the user. The main method must print “Incorrect license format”, if method

`computeChecksum` returns -1, otherwise it should print the value returned by method `computeChecksum.`

8. The figure below demonstrates the format of a file containing strings with *last name, age* and *score* information separated by spaces. Note: the first number on a line represents the age.

```
Williams   9 15
Collins    10 4
Zaldastani 9 12
Silverman  10 23
Lerner      9 40
Vindig     9 11
Soltan     15 34
Violetta   12 43
Germon     23 18
```

Develop a **static** method `winnerInAgeGroup()` that must be passed three parameters: a file name and two integer numbers, representing low and high number for the age group.

The method should **return** the last name of a person with the age that fits within the range specified by parameters, whose score is the highest score of all people in that range. For instance, given the file above and range values 9 and 11, the method should return `Lerner` (as a String), since Lerner is a 9-year old with the greatest score (40) among all people of ages 9,10, 11.

If there is no person in the list within the age group designated by the parameter, the method should return an empty string. Assume all input is entered in valid form and that there are no two people with the same score.