Java Programming

U Hou Lok

Department of Computer Science and Information Engineering,
National Taiwan University

Java 273
22 Aug.–2 Sep., 2016

```
1  class Lecture5 {
2
3                    "Arrays"
4
5  }
```

Arrays

An array stores a large collection of data which is of the same type.

```
1  ...
2          // assume the size variable exists above
3          T[] x = new T[size];
4          // this creates an array of T type referenced by x
5  ...
```

- **T** can be any data type.
- This statement comprises two parts:
  - ▶ Variable declaration
  - ▶ Creating an array

Variable Declaration

- In the left-hand side of the assignment operator, it is a declaration for an array variable, which does not allocate real space for the array.
- In reality, this variable occupies only a certain space for the reference to an array.[1]
- If a reference variable does not refer to an array, the value of the variable is null.[2]
- So you cannot assign elements to this array variable unless it has already been created.

---

[1] Recall the stack and the heap in the memory layout.

[2] Moreover, this holds for any reference variable. For example, the **Scanner** type.

Creating A Real Array

- All arrays of Java are objects.
- So the new operator returns the reference after creating an array object.
- The type of reference variables must be compatible to that of the array object.
- The variable *size* must be a positive integer for the number of elements.
- Note that the size of an array cannot be changed after the array is created.[3]

---

[3]You can try the **ArrayList** class. See any textbook for data structures.

Arrays and Memory

- Consider this array whose elements are all int type:

```java
int[] x = new int[3];
```


0x000abc26   0x000abc30   0x000abc34
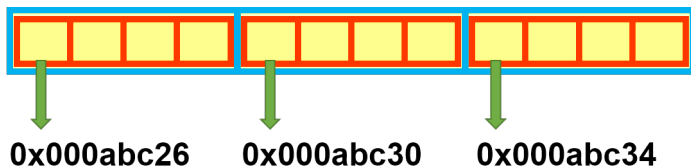
- The array is allocated contiguously in the memory.
- Note that the arrays (in Java) are zero-based indexing. (Why?)
- So we have x[0], x[1], and x[2].

Array Initializer

Arrays can be initialized when they are declared.

- When an array is created, the elements are assigned the default value:
    - ▶ 0 for the numeric primitive data types
    - ▶ \u0000 for char type
    - ▶ false for boolean type
- An array also can be initialized by enumerating all the elements in a list.
- For example,

```java
int[] x = {1, 2, 3};
```

- Note that there is no need to use new if enumeration is used.

Processing Arrays

> When processing array elements, we often use a for loop.

- Since the size of the array is known, it is natural to use a for loop to manipulate the array.
- For all arrays, they have a field called **length** which records the size of this array.
  - For example, use x.length to get the size of x.

Examples

Initializing arrays with input values

```
1 ...
2          // let x be an integer array with a certain size
3          for (int i = 0; i < x.length; ++i) {
4              x[i] = input.nextInt();
5          }
6 ...
```

Initializing arrays with random values

```
1 ...
2          for (int i = 0; i < x.length; ++i) {
3              x[i] = (int) (Math.random() * 10);
4          }
5 ...
```

Displaying arrays

```
1  ...
2          for (int i = 0; i < x.length; ++i) {
3              System.out.printf("x[%d] = %d\n", i, x[i]);
4          }
5  ...
```

Summing all elements

```
1  ...
2          int sum = 0;
3          for (int i = 0; i < x.length; ++i) {
4              sum += x[i];
5          }
6  ...
```

Finding the extreme values

```
1  ...
2          int max = x[0];
3          int min = x[0];
4          for (int i = 1; i < x.length; ++i) {
5                  if (max < x[i]) max = x[i];
6                  if (min > x[i]) min = x[i];
7          }
8  ...
```

- How about the location of the extreme values?

Shuffling

```
 1   ...
 2           // Assume that x is an integer array.
 3           for (int i = 0; i < x.length; ++i) {
 4               int j = (int) (Math.random() * x.length);
 5               // swap
 6               int tmp = x[i];
 7               x[i] = x[j];
 8               x[j] = tmp;
 9           }
10   ...
```

- How to swap values of two variables without tmp?
- However, this simple (or naive) algorithm is biased.[4]

---

[4]See https://blog.codinghorror.com/the-danger-of-naivete/.

Exercise[5]

Generate 4 distinct integers randomly ranging from 0 to 9.

- Consider this algorithm.
  - ▸ Generate the 1st random integer and the 2nd random integer.
  - ▸ Then check if the second one is identical to the first integer.
  - ▸ If so, redraw and check again.
  - ▸ If not, then generate the next random integer and repeat the previous steps until the 4 distinct integers are set.
- Argue the time complexity for this algorithm.


[5]Thanks to a lively discussion on January 24, 2016.

Cloning Arrays

- In practice, one might duplicate an array for some reason.
- One could attempt to use the assignment statement (=), for example,

```
...
T[] A = {...}; // some array
T[] B = A;
...
```

- However, this is impossible to make two distinct arrays.
- Recall that arrays are objects and the array variables are simply references to these objects.
- So you actually copy the memory address to another reference variable!

- Moreover, all the reference variables share this property!
- For example,
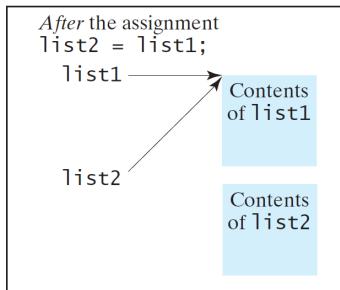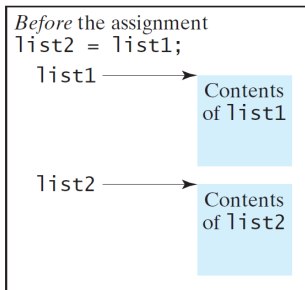
- Use a loop to copy individual elements one by one.

```
1   ...
2   int[] A = {2, 1, 3, 5, 10};
3   int[] B = new int[A.length];
4   for (int i = 0; i < A.length; ++i) {
5   B[i] = A[i];
6   }
7   ...
```

- Use the static *arraycopy* method in the **System** class.

```
1   ...
2   int[] A = {2, 1, 3, 5, 10};
3   int[] B = new int[A.length];
4   System.arraycopy(A, 0, B, 0, A.length);
5   ...
```

for-each Loops[6]

- A for-each loop is designed to iterate over a collection of objects, such as an array, in strictly sequential fashion, from start to finish.
- For example,

```
1   ...
2   int[] A = {...}; // some T type array
3   for (T x: A) {
4   // body
5   }
6   ...
```

- Note that the type **T** should be compatible to the element type of A.

---

[6]Beginning with Java SE 5. Now we have Jave SE 8.

Example

```
1   ...
2   int[] A = {1, 2, 3};
3   int sum = 0;
4   for (int i = 0; i < A.length; ++i)
5   sum += A[i];
6   ...
```

- Not only is the syntax streamlined, but it also prevents boundary errors.

```
1   ...
2   int[] A = {1, 2, 3};
3   int x = 0;
4   for (int x: A)
5   sum += x;
6   ...
```

Exercise

Deck of Cards

Write a program which picks first 5 cards at random from a deck of 52 cards.

- 4 suits: Spade, Heart, Diamond, Club
- 13 ranks: $2, \ldots, 10$, J, Q, K, A
- Labeling 52 cards by $0, 1, \cdots, 51$ by a certain rule
- Shuffle the numbers
- Pick up the first 5 cards

Arrays Class

- The **Arrays** class contains useful methods for common array operations such as sorting and searching.
- For example,

```java
import java.util.Arrays;

...
int[] A = {5, 2, 8};
Arrays.sort(A); // sort the whole array

char[] B = {'A', 'r', 't', 'h', 'u', 'r'};
Arrays.sort(B, 1, 3); // sort the array partially
...
```

Selection Sort

```
1    ...
2    // selection sort
3    for (int i = 0; i < A.length; i++) {
4    int k = i;
5    for (int j = i; j < A.length; j++) {
6    if (A[k] > A[j]) k = j;
7    }
8    int tmp = A[k];
9    A[k] = A[i];
10   A[i] = tmp;
11   }
12   ...
```

- Time complexity: $O(n^2)$
- You can find more sorting algorithms.[7]

---

[7]See http://www.sorting-algorithms.com/

Example: Linear Search

> Write a program which searches for the index associated with the key value.

- The linear search approach compares the key with each element in the array sequentially.
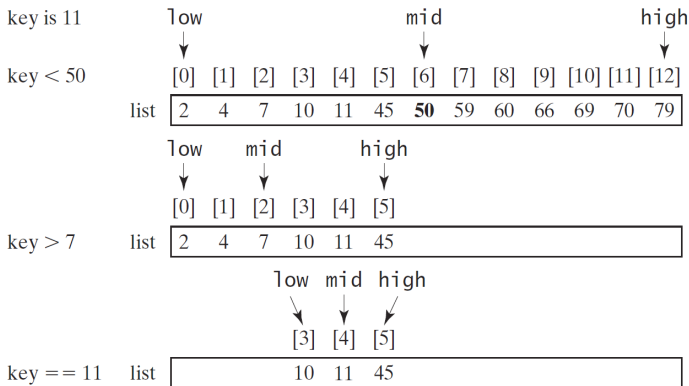
```java
1  ...
2  public static void main (String[] args) {
3  int[] A = new int[10];
4  for (int i = 0; i < A.length; i++) {
5  A[i] = (int) (Math.random() * 10);
6  System.out.printf("[%2d]: %d\n", i, A[i]);
7  }
8
9  Scanner in = new Scanner(System.in);
10 System.out.println("Enter a key number? ");
11 int key = in.nextInt();
12 in.close();
13
14 int index = -1;
15 for (int i = 0; i < A.length; i++) {
16 if (A[i] == key) {
17 System.out.printf("%2d", i);
18 }
19 }
20 }
21 ...
```

- Time complexity: $O(n)$

Alternative: Binary Search

key is 11

key < 50

```
        low                    mid                    high
         ↓                      ↓                      ↓
        [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9] [10] [11] [12]
list     2    4    7   10   11   45   50   59   60   66   69   70   79
```

```
        low       mid       high
         ↓         ↓         ↓
        [0]  [1]  [2]  [3]  [4]  [5]
```

key > 7

```
list     2    4    7   10   11   45
```

```
                   low  mid high
                    ↓    ↓   ↓
                   [3]  [4]  [5]
```

key == 11

```
list              10   11   45
```

- Time complexity: $O(\log n)$
- Overall time complexity (sorting + searching): still $O(\log n)$?

```
1    ...
2    int index = -1;
3
4    int high = A.length - 1, low = 0, mid;
5    while (high > low) {
6    mid = (high + low) / 2;
7    if (A[mid] == keyValue) {
8    index = mid;
9    break;
10   } else if (A[mid] > keyValue)
11   high = mid - 1;
12   else
13   low = mid + 1;
14   }
15
16   if (index > -1)
17   System.out.printf("%d: %d\n", keyValue, index);
18   else
19   System.out.printf("%d: does not exist\n", keyValue);
20   ...
```

Data Structures In A Nutshell

- A data structure is a particular way of organizing data in a program so that it can be used efficiently.
- Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computational complexity of those operations.
- In comparison, a data structure is a concrete implementation of the specification provided by some ADT.
- Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.[8]

---

[8]See http://bigocheatsheet.com/.

Beyond 1-Dimensional Arrays

- 2D or high-dimensional arrays are common, say a photo which is represented by 2D arrays.
- For example, we can have a 2D **T** array with 4 rows and 3 columns:

```
1    ...
2    int rowSize = 4; // row size
3    int colSize = 3; // column size
4    T[][] x = new T[rowSize][colSize];
5    ...
```

	[0][1][2][3][4]
[0]	0 0 0 0 0
[1]	0 0 0 0 0
[2]	0 0 0 0 0
[3]	0 0 0 0 0
[4]	0 0 0 0 0

```
matrix = new int[5][5];
```

(a)

	[0][1][2][3][4]
[0]	0 0 0 0 0
[1]	0 0 0 0 0
[2]	0 7 0 0 0
[3]	0 0 0 0 0
[4]	0 0 0 0 0

```
matrix[2][1] = 7;
```

(b)

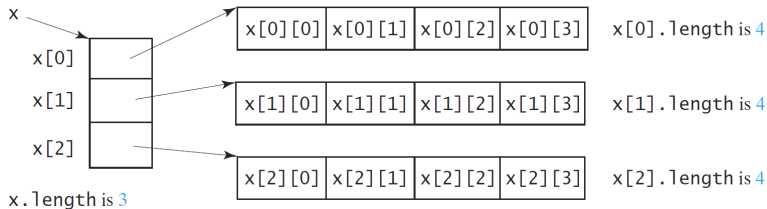|	[0][1][2]
[0]	1 2 3
[1]	4 5 6
[2]	7 8 9
[3]	10 11 12

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(c)

- Case (c) shows that we can create a 2D array by enumeration.

Reality


```
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
```

Example[9]

```
1    ...
2    int[][] x = {{1, 2, 3}, {4, 5}, {6}};
3
4    for (int i = 0; i < x.length; i++) {
5    for (int j = 0; j < x[i].length; j++) {
6    System.out.printf("%2d", x[i][j]);
7    }
8    System.out.println();
9    }
10
11   // alternative: for-each loop
12   for (int[] y: x) {
13   for (int z: y) {
14   System.out.printf("%2d", z);
15   }
16   System.out.println();
17   }
18   ...
```

---

[9]Thanks to a lively discussion on January 31, 2016.

Exercise: Matrix Multiplication

Write a program which determines $C = A \times B$ for the input matrices $A_{m \times n}$ and $B_{n \times q}$ for $m, n, q \in \mathbb{N}$.

- You may use the formula

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where a_{ik}, $i = 1, 2, \ldots, m$ is a shorthand for A and b_{kj}, $j = 1, 2, \ldots, q$ for B.

- Time complexity: $O(n^3)$ (Why?)

Data Structures In A Nutshell

- A data structure is a particular way of organizing data in a program so that it can be used efficiently.

- Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computational complexity of those operations.

- In comparison, a data structure is a concrete implementation of the specification provided by some ADT.

- Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.[10]

---

[10]See http://bigocheatsheet.com/.

Beyond 1-Dimensional Arrays

- 2D or high-dimensional arrays are common, say a photo which is represented by 2D arrays.
- For example, we can have a 2D **T** array with 4 rows and 3 columns:

```
    ...
    int rowSize = 4; // row size
    int colSize = 3; // column size
    T[][] x = new T[rowSize][colSize];
    ...
```

```
[0][1][2][3][4]
[0]  0  0  0  0  0
[1]  0  0  0  0  0
[2]  0  0  0  0  0
[3]  0  0  0  0  0
[4]  0  0  0  0  0
matrix = new int[5][5];

(a)
```

```
[0][1][2][3][4]
[0]  0  0  0  0  0
[1]  0  0  0  0  0
[2]  0  7  0  0  0
[3]  0  0  0  0  0
[4]  0  0  0  0  0
matrix[2][1] = 7;

(b)
```
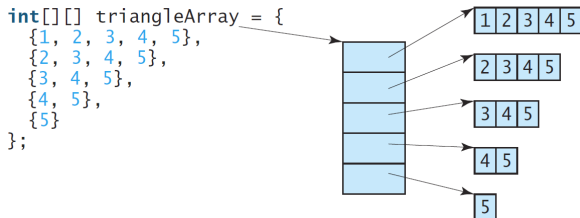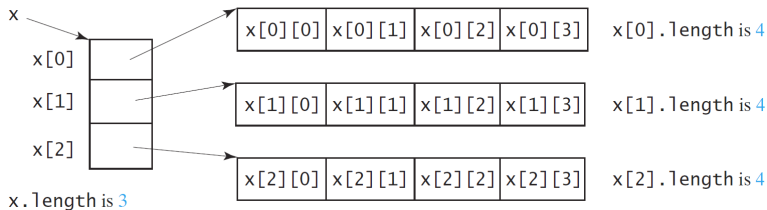
```
[0][1][2]
[0]  1  2  3
[1]  4  5  6
[2]  7  8  9
[3] 10 11 12
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};

(c)
```

- Case (c) shows that we can create a 2D array by enumeration.

Reality

Example[11]

```
1   ...
2   int[][] x = {{1, 2, 3}, {4, 5}, {6}};
3
4   for (int i = 0; i < x.length; i++) {
5   for (int j = 0; j < x[i].length; j++) {
6   System.out.printf("%2d", x[i][j]);
7   }
8   System.out.println();
9   }
10
11  // alternative: for—each loop
12  for (int[] y: x) {
13  for (int z: y) {
14  System.out.printf("%2d", z);
15  }
16  System.out.println();
17  }
18  ...
```

---

[11]Thanks to a lively discussion on January 31, 2016.

Exercise: Matrix Multiplication

> Write a program which determines $C = A \times B$ for the input matrices $A_{m \times n}$ and $B_{n \times q}$ for $m, n, q \in \mathbb{N}$.

- You may use the formula

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

  where a_{ik}, $i = 1, 2, \ldots, m$ is a shorthand for A and b_{kj}, $j = 1, 2, \ldots, q$ for B.
- Time complexity: $O(n^3)$ (Why?)

```
1    class Lecture6 {
2
3    "Methods"
4
5    }
6
7    // keywords:
8    return
```

Methods[13]

- Methods can be used to define reusable code, and organize and simplify code.
- The idea of function originates from math, that is,

$$y = f(x),$$

where x is the input parameter[12] and y is the function value.
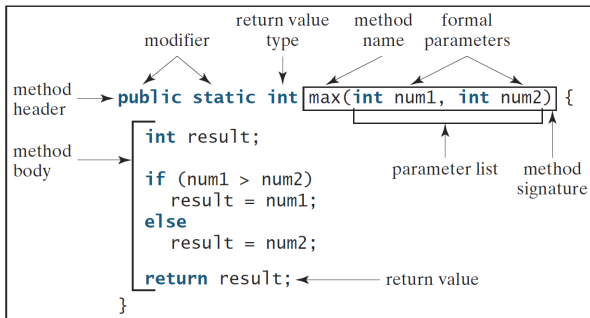- In computer science, each input parameter should be declared with a specific type, and a function should be assigned with a return type.

---

[12]Recall the multivariate functions. The input can be a vector, say the position vector (x, y, z).

[13]Aka procedures and functions.

Example: max

Define a method

Invoke a method


```
                    return value  method   formal
         modifier       type       name   parameters

method
header  →  public static int  max(int num1, int num2)  {

method
body    →    int result;

             if (num1 > num2)
               result = num1;
             else
               result = num2;

             return result;  ←——— return value
         }
```

parameter list   method signature

```
int z = max(x, y);

         actual parameters
            (arguments)
```

```
1    ...
2    modifier returnType methodName(listOfParameters) {
3    // method body
4    }
5    ...
```

- So far, the *modifier* could be static and public.
- The *returnType* could be primitive types, reference types[14], and void.[15]
- The *listOfParameters* is the input of the method, separated by commas if there are multiple items.
- Note that a method can have no input.[16]
- The method name and the parameter list together constitute the method signature.[17]

---

[14] A method can return an object.

[15] Recall that a void method does not return a value.

[16] For example, **Math.random**().

[17] It is the key to the method overloading. We will see it soon.

More Observations

- There are alternatives to the method max():

```
...
public static int max(int x, int y) {
if (x > y) {
return x;
} else {
return y;
}
}
...
```

```
...
public static int max(int x, int y) {
return x > y ? x : y;
}
...
```

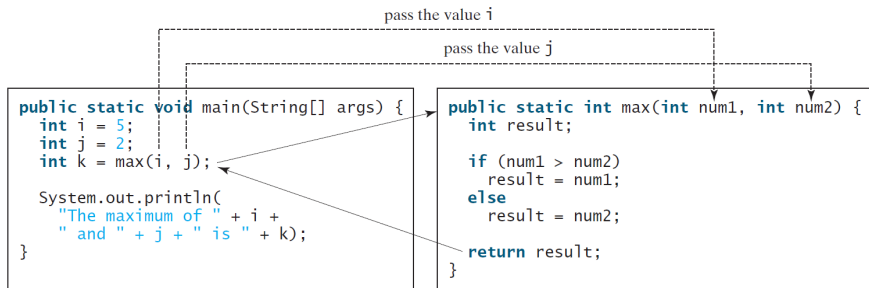"All roads lead to Rome."
– Anonymous

""
–

The return Statements

- The return statement is the end point of the method.
- A callee is a method invoked by a caller.
- The callee returns to the caller if the callee
  - completes all the statements (w/o a return statement, say main());
  - reaches a return statement;
  - throws an exception (introduced later).
- Note that the return statement is not necessarily at the bottom of the method.[18]
- Once one defines the return type (except void), the method should return a value or an object of that type.

---

[18]Thanks to a lively discussion on November 22, 2015.

- However, your method should make sure that the return statement is available for all conditions.
- For example,

```
1    ...
2    static int fun1() {
3    while (true);
4    return 0; //unreachable code
5    }
6
7    static int fun2(int x) {
8    if (x > 0) {
9    return x;
10   }
11   // what if x < 0?
12   }
13   ...
```

Method Invocation


```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum of " + i +
        " and " + j + " is " + k);
}
```

pass the value i

pass the value j

```java
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

- Note that the input parameters are sort of variables declared within the method as placeholders.
- When calling the method, one needs to provide arguments, which must match the parameters in order, number, and compatible type, as defined in the method signature.

- In Java, method invocation uses pass-by-value.
- When the callee is invoked, the program control is transferred to the caller.
- For each invocation of methods, OS creates an frame which stores necessary information, and the frame is pushed in the call stack.
- The callee transfers the program control back to the caller once the callee finishes its job.
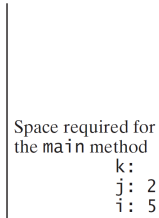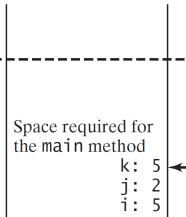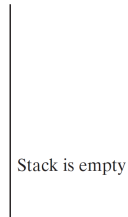
(a) The main method is invoked.
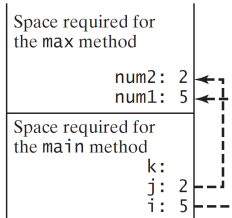
(b) The max method is invoked.
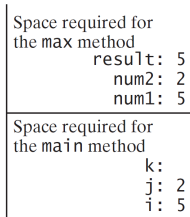
(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.

Variable Scope

- The variable scope is the range where the variable can be referenced.
- Variables can be declared in class level, method level, and loop level.
- In general, a balanced curly brackets defines a particular scope.
- One can declare variables with the same name in different levels of scopes.
- Yet, one cannot declare the variables with the same name in the same scope.

Example

```java
public class ScopeDemo {

    static int i = 1; // class level

    public static void main(String[] args) {
        System.out.printf("%d\n", i); //output 1
        int i = 2; // method level; local
        i++;
        System.out.printf("%d\n", i); // output 3
        p();
        System.out.printf("%d\n", i); // output ?
    }

    static void p() {
        i = i + 1;
        System.out.printf("%d\n", i); // output ?
    }
}
```

Exercise

Write a program which allows the user to enter the math grades one by one (-1 to exit), and outputs a histogram.

```
Enter (-1 to exit): 40
Enter (-1 to exit): 90
Enter (-1 to exit): 86
Enter (-1 to exit): 56
Enter (-1 to exit): 78
Enter (-1 to exit): 64
Enter (-1 to exit): -1
End of Input...
Total: 6
 90 ~ 100: *
 80 ~  89: *
 70 ~  79: *
 60 ~  69: *
  0 ~  59: **
```

```java
1    ...
2    int[] hist = new int[5];
3    // hist[0]: the counter for 90 ~ 100
4    // hist[1]: the counter for 80 ~ 89
5    // ...
6    // hist[4]: the counter for 0 ~ 59
7
8    Scanner in = new Scanner(System.in);
9    int inputValue;
10   do {
11   do {
12   System.out.println("Enter x? (-1 to exit)");
13   inputValue = in.nextInt();
14   } while (inputValue != -1 && (inputValue < 0 || inputValue > 100));
15   if (inputValue >= 90) ++hist[0];
16   else if (inputValue >= 80) ++hist[1];
17   else if (inputValue >= 70) ++hist[2];
18   else if (inputValue >= 60) ++hist[3];
19   else if (inputValue >= 0) ++hist[4];
20   else System.out.println("End of input.");
21   } while (inputValue != -1);
22   in.close();
23
24   int total = 0;
25   for (int i = 0; i < hist.length; i++) {
```

```
26      total += hist[i];
27      }
28      System.out.printf("Total: %d\n", total);
29
30      int[] tick = {0, 60, 70, 80, 90, 101};
31      for (int i = 0; i < 5; i++) {
32      System.out.printf("%3d ~ %3d: ", tick[i], tick[i + 1] − 1);
33      for (int j = 0; j < hist[i]; j++) {
34      System.out.printf("*");
35      }
36      System.out.printf("\n");
37      }
38      ...
```

- Shall we rearrange the codes in order to reduce the complexity of the main method?