Iterarions (or Loops)

Lecture 4

Dr Jonas Lundberg, office B3024

`Jonas.Lundberg@lnu.se`

`Slides are available in Moodle`

November 3, 2019

Control Statements

By using **control statements** the program can choose one execution path out of several possible options or it can repeat a sequence of statements several times.

- Until Now:
  1. Sequential execution (from the top and downwards)
  2. If-statement ⇒ Choose one out of several possible options
  3. Method calls ⇒ jump to another location in the code (and back)
- Control Statements:
  - **Selective** statements: Choose one execution path out of several possible options
  - In Java: if- or switch-statements
  - **Iterative** (or **loop**) statements: Repeat a sequence of statements several times
  - In Java: while-, do-, or for-statements

Agenda – Control Statements

- Iterative statements
  - while statements
  - do statements
  - for statements
- Nestled Statements
- Solving problems using if, while, and for

Reading instructions
Liang: Chapter 5

Assignment tasks: Assignment 2 (plenty of exercises!)

Introduction to `while`

Problem: Find smallest K such that $1 + 2 + 3 + 4 + ... + K > 10000$

```
public static void main(String[] args) {
   int K = 0, sum = 0;
   while (sum <= 10000) {
      K++;
      sum = sum + K;          // 1+2+3+4+ ....
   }
   System.out. println ("K = "+K+ " ==> 1+2+3+ ... K = "+sum);
}
```

Output: K = 141 ==> 1+2+3+ ... K = 10011

While Statement

```
while ( boolExpr ) {
  ...
}
next statement;
```

- The while statement will be repeated until `boolExpr` has the value `false`
- That means that the statement block inside the while statement might be executed several times.
- To be used when we don't know *how many* times the loop should be executed, we just know under *which condition* it should terminate.
- Alternative: The statement block can be replaced by a single statement

```
while ( boolExpr )
   statement;
next statement;     // not included in the while statement
```

while **Example:** MeanValue.java

```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int sum = 0, count = 0, value = -1;

    while (value != 0) {
        count++;                    // iteration count
        System.out.print("Give an integer (zero to stop): ");
        value = scan.nextInt();
        sum = sum + value;    // sum of all input values
    }
    count--;  // Don't count last input (zero)

    System.out.println("\nFinal sum: "+sum);
    System.out.println("Number of integers: " +count);
    System.out.println("Mean value: "+(double)sum/count);
}
```

Read multiple data from keyboard

```
Scanner scan = new Scanner(System.in);
System.out.print("Provide any number of integers and stop with an X: ");
while (scan.hasNextInt()) {     // Do we have any more integers?
    int num = scan.nextInt();
    System.out.print(num + " ");
}
System.out.println("\nFinal input: "+ scan.nextLine());
```

Execution

```
Provide any number of integers and stop with an X: 11 22 33 44 hello
11 22 33 44
Final input: hello
```

Hence, `scan.hasNextInt()` $\Rightarrow$ `true` while we are reading integers.
Advantage: We can read many integers in one line and interupt the process by providing a non-integer

The do-Statement

The do statement is closely related to the while statement

```
do {
  ...
} while ( boolExpr );   // Note the final semicolon
```

- The do statement is repeated until `boolExpr` gets the value `false`
- do or while?
  - Repeat **one** or several times ⇒ do statement
  - Repeat **zero** or several times ⇒ while statement

Example: Continue (Y/N)?

```
do {
     ...

    System.out.print(''Continue (Y/N)?:  '');
    char r = scanner.next().charAt(0);   // First char in next word
} while ( r  == 'y' || r == 'Y' );
```

The keywords `break` and `continue`

- `break` and `continue` is used to jump out of a loop at an arbitrary position.
- Example

```
while ( boolExpr ) {
  ...
  if (...)
    break;  // End the loop, jump to next statement

  if (...)
    continue; // End the iteration, jump to while ( boolEx
  ...
}
next statement;
```

- `break` and `continue` are considered to make the code more difficult to understand ⇒ use them with care!

Introduction to `for`

Problem: Print odd numbers between 5 and 95

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

    for (int i = 5; i <= 95; i = i+2) {
        System.out.println( i );
    }
}
```

- `int i = 5` ⇒ i starts with valuet 5
- `i <= 95` ⇒ loop as long as `i <= 95`
- `i = i+2` ⇒ icrease value of i with 2 in the end of each turn

For Statement

```
for ( init; boolExpr; change ) {      for ( int i=0; i<100; i++) {
  ...                                   ...
}                                     }
next statement;
```

- The `for` statement start with an `initialization` (e.g. `int i = 0`)
  Usually a **counter** variable is declared and assigned a start value.
- The `for` statement is repeated until `boolExpr` gets the value `false`
  Usually this is a condition regarding the value of the counter (e.g. `i < 100`)
- Each iteration ends with `change`
  Usually the counter is incremented or decremented. (e.g. `i++` or `i=i+2`)
- Problem: Add all odd numbers between 1 and 99

```
int sum = 0
for (int i=99; i>0; i=i-2 ) {    // i = 99,97,95,93, ....
   sum = sum + i;
}
System.out.println("99+97+95+ ... +5+3+1 = "+sum);
```

- `for` statements are used when we know how many iterations to be executed
- Note: the counter `i` can only be used inside the loop.

for **Example:** MultiplicationTable.java

```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter positive integer N: ");
    int N = scan.nextInt();

    System.out.println("\n****** Multiplication Table for N ********");
    for (int i=0; i<=N; i++) { // i = 0,1,2,3,..., N
        int n = i * N;
        System.out.println(i+" * "+N+" = "+n);
    }
}
```

Example Execution:

```
Enter positive integer N: 4
****** Multiplication Table for N ********
0 * 4 = 0
1 * 4 = 4
2 * 4 = 8
3 * 4 = 12
4 * 4 = 16
```

10 Minute Break!

ZZZZZZZZZZZZZZZZZZZZZZZZZZ

Nestled Statements

- Understanding each control statement by itself is rather easy
- Solving problem requiring only one such statement is also often rather easy
- However, many problems require multiple nestled control statements
- **Nestled** ⇒ statements inside other statement

```
if (n > 0) {
    if ( n%2=0) {
        ...
    }
    else {
        ...
    }
else {
    for (int i=n; i<=0;i++) {
        ...
    }
}
```

- Solution with nestled statements ⇒ much harder ⇒ much training needed

Example: Count A

Write a program `CountA.java` that reads a string from the keyboard and then prints how many 'a' and 'A' the string contains. An example of what an execution might look like:

```
Provide a line of text: All cars got the highest safty grading A.
Number of 'a': 3
Number of 'A': 2
```

Sketch of a Solution

1. Read a line of text ⇒ `String text`
2. For each `char c` in `text`
    - if `c = 'A'` ⇒ `nA++`
    - else if `c = 'a'` ⇒ `na++`
3. Print result ⇒ Print `nA` and `na`

Hint: I should have waited with 1 (read line of text) until the end. Why?

Solution - Count A

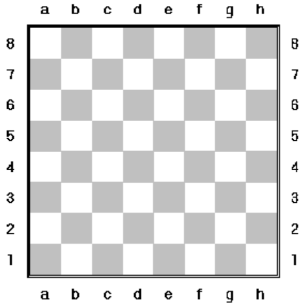
```
public static void main(String[] args) {
    // Read input text
    Scanner scan = new Scanner(System.in);
    System.out.print("Provide a line of text: ");
    String text = scan.nextLine();

    // Traverse string and count A:s and a:s
    int numberOfAs = 0;
    int numberOfas = 0;
    for (int i=0; i<text.length(); i++) {        // Iterate over
        char ch = text.charAt(i);                 // string characters
        if (ch == 'A')
            numberOfAs = numberOfAs + 1;
        else if (ch == 'a')
            numberOfas = numberOfas + 1;
    }

    // Present result
    System.out.println("Number of \'A\': "+numberOfAs);
    System.out.println("Number of \'a\': "+numberOfas);
}
```

Linnæus University

Exercise: Color of Chess Square



Each square in a chess board in identified by a letter (a-h) and an integer (1-8). They are typically refered to as c3 or f5. Write a program **SquareColor.java** that reads a square identifier (e.g. c5) from the user and prints the color (Dark or Light).

Solution: Color of Chess Square

```
// User instructions + Reading square identifier
System.out.print("Enter Chess Square identifier (e.g. e5): ");
Scanner scan = new Scanner(System.in);
String text = scan.nextLine();          // e.g. "e5"
char letter = text.charAt(0);           // 'e'
char digitChar = text.charAt(1);        // '5'
int digit = Character.getNumericValue(digitChar);   // char-to-int

if (digit % 2 == 0) {// Even row ==> 2,4,6,8
    if (letter =='a' || letter =='c' || letter =='e' || letter =='g')
        System.out.println("The square is light");
    else
        System.out.println("The square is dark");
}
else {// Odd row ==> 1,3,5,7
    if (letter =='a' || letter =='c' || letter =='e' || letter =='g')
        System.out.println("The square is dark");
    else
        System.out.println("The square is light");
}
```

Exercise: Stupid Encryption

A very simple (stupid?) way to encrypt a text would be to just shift each letter one step in the alphabet. That is, replace all letters in the text with the next letter in the alphabet

```
a --> b, b --> c,    ...    , y --> z, z --> a
A --> B, B --> C,    ...    , Y --> Z, Z --> A
```

All non-letters, for example digits, ? ,!, %, and whitespace, are left unchanged.

Exercise: Write a program `StupidEncryption.java` that reads a line of text from the user and presents an encrypted version of the text according to the encryption method outlined above. An execution might look like this:

```
Provide a line of text: Was it a rat I saw?
Encrypted Text: Xbt ju b sbu J tbx?
```

Solution: Stupid Encryption

```
        System.out.print("Provide a line of text: ");
        Scanner scan = new Scanner(System.in);
        String text = scan.nextLine();
//      String text = "abcdefghijklmnopqrstuvxyz ABCDEFGHIJKLMNOPQRSTUVXYZ!?.

        StringBuilder buf = new StringBuilder();
        for (int i=0; i<text.length(); i++) {
            char c = text.charAt(i);
            if ( !Character.isLetter(c) )   // Non-letter ==> do nothing
                buf.append(c);
            else if (c == 'z')              // Special case 'z'
                buf.append('a');
            else if (c == 'Z')
                buf.append('A');                      // Special case 'z'
            else {        // Default: Convert to ASCII, add 1, convert back to char
                int ascii   = (int) c;      // ASCII Value
                char nextChar = (char) ( ascii +1);
                buf.append(nextChar);
            }
        }
        System.out.println("Encrypted Text: "+buf.toString());
```

Programming: Old Exam Exercise

Write a Java program Square.java that first reads any integer (higher than or equal to 3) from the keyboard and then prints a non-filled square of the type presented below. An execution might look like this:

```
Provide an integer 3 or higher: 5
The square for number 5
  *****
  *   *
  *   *
  *   *
  *****
```

An error message should be given, and the program should terminate, if the user-provided integer value is below three.