Java: Strings and Methods

Instructor: Nihshanka Debroy
Recap of Loops
Anatomy of a for loop

```java
int i;
for(i=1;i<=10;i++)
{
    System.out.println("Hello World");
}
```

This can be interpreted as:

Start by setting `i=1`.
As long as (`i<=10`), keep doing this:

- More than one statement in loop: use curly braces
- `i=1` (initialization step)
- `i<=10` (condition that we check each time)
- `i++` (increase i by 1), is the update/increment step

Increase i by 1
Anatomy of a while loop

```java
int i = 1;
while (i <= 10) {
    System.out.println("Hello World");
    i = i + 1;
}
```

This can be interpreted as:

- Start by setting $i = 1$.
- As long as ($i \leq 10$), keep doing this:
  ```java
  print "Hello World"
  Increase i by 1
  ```

- $i = 1$ (initialization step)
- $i \leq 10$ (condition that we check each time)
- $i = i + 1$ (increase i by 1), is the update/increment step
do while loop

i=1;
do {
    System.out.print("This is a do while loop");
    i=i+1;
} while ( i<10 );

This can be interpreted as:
Start by setting i=1.
Keep doing this:
{
    print "This is a do while loop"
    Increase i by 1
} As long as i <10

**Important:** do while loop gets executed at least once, as opposed to a while loop (because in a do while loop, the condition gets checked only after the first iteration)
Sum of positive integers

- Analyze the problem
  - Input
    - n a positive integer
  - Output
    - sum sum of all positive integers <= n
  - How to find the sum??
    - sum = 1+2+3+ ... +n
    - initialize sum=0
    - let k loop over the values [1, n]
    - compute sum=sum+k at each iteration of loop
Sum of positive integers

- Design an algorithm to solve the problem

1. Get input value for \( n \)
2. Compute sum of integers 1 through \( n \)

```c
sum=0;
for(k=1; k<=n; k++)
{
    sum=sum+k;
}
```

3. Return output value \( sum \)
Factorial

- Analyze the problem
  - Input
    - n a positive integer
  - Output
    - fn n!, the factorial of n
- How to find the factorial??
  - fn = 1*2*3* ... *n
  - initialize fn=1
  - let k loop over the values [2, n]
  - compute fn=fn*k at each iteration of loop

why don’t we set fn=0?
Factorial

- Design an algorithm to solve the problem

1. Get input value for $n$
2. Compute product of integers 2 through $n$

```java
fn=1;
for(k=2; k<=n; k++)
{
    fn=fn*k;
}
```

3. Return output value $fn$

→ what happens when $n=1$?
Java String Class

- String class represents character strings

  ```java
  String name1 = "John";
  String name2 = "Jennifer";
  String course = new String("Lecture11");
  String shorterName = name.substring(0,3);  // which is "Lec"
  ```

- Strings can be concatenated (added together) by using the + sign

  ```java
  String names = name1 + " " + name2;
  ```

- Testing for equality

  ```java
  name1.equals("John");  // returns true
  name2.equals("John");  // returns false
  ```
Java Escape Sequences

- `\n`: newline
- `\b`: backspace
- `\t`: tab
- `\r`: carriage return
- `\"`: double quote
- `\'`: single quote
- `\\`: backslash
Subroutines

- Set of instructions to perform a particular computation
  - subproblems of more complex problems
  - repeated computation (e.g. different inputs)
  - may be used in solving other problems
- Also called subprograms, methods, functions, procedures
- Subroutines are named
- Subroutines have zero or more parameters
  - list (possibly empty) of input values and their types
- Subroutines have return types
Syntax

```
return-type subroutine-name( parameter-list )
{
    … statements …
}
```

- The *return-type* specifies the data type of the output
- The *parameter-list* is a comma-separated list of the input variables and their types
  - each list item has the form *input-type variable-name*
- A program refers to the subroutine by *subroutine-name*
- The body consists of the *statements* between `{ }`
A subroutine to add two integers

```c
int add(int x, int y)
{
    return x+y;
}
```
Minimum of two integers - algorithm

- Problem: Find the minimum of two integers
- Algorithm:
  1. Get input values for $x$ and $y$
  2. Compute minimum value
     
     ```
     if(x < y)
     min = x;
     else
     min = y;
     ```
     
  3. Return output value $min$
Minimum of two integers - subroutine

```c
int minimum(int x, int y)
{
    int min;
    if(x < y)
        min=x;
    else
        min=y;
    return min;
}
```

Variables used inside the subroutine must be declared.

Variables cannot have the same name as the subroutine.
int sum_integers(int n)
{
    int k;
    int sum=0;
    for(k=1; k<=n; k++)
    {
        sum=sum+k;
    }
    return sum;
}
int factorial(int n)
{
    int k;
    int fn=1;
    for(k=2; k<=n; k++)
    {
        fn=fn*k;
    }
}
Top-down design

- Top-down design
  - original problem is divided into simpler independent subproblems
  - successive refinement: these subproblems may require further division into smaller subproblems
  - continue until all subproblems can be easily solved or you already have a method to solve them
Top-down design

Compute minimum of $x$, $y$, and $z$

- Compute minimum of $x$ and $y$
  - SUBPROBLEM1

- Compute minimum of solution to SUBPROBLEM1 and $z$
  - SUBPROBLEM2
Calling subroutines

- A subroutine \texttt{call} statement has the following syntax:

  \texttt{subroutine-name( parameters );}

- Calling a subroutine executes the statements in the body of the called subroutine using the specified \texttt{parameters}
- \texttt{parameters} is a comma-separated list of input values
  - the input values must be of the same type as those in the subroutine’s parameter list
- The subroutine call statement may be part of an assignment statement or an return statement
Minimum of three integers

- We can call our `minimum` subroutine twice to compute the minimum of three integers

```c
int min_xy, min_xyz;
min_xy = minimum(x, y);
min_xyz = minimum(min_xy, z);
```
public class meth {

    //@param args

    public static void main(String[] args) {
        int x, y, z, a, b;
        x=4;
        y=3;
        z=5;
        a = min(x,y);
        b = min(a,z);
        System.out.println(b);
    }

    public static int min(int x, int y) {
        if(x<y)
            return x;
        else
            return y;
    }
}
Syntax errors

- The following Java subroutine computes the inclusive sum between two integers. Find all the syntax errors.

```java
int sumBetween( x, y )
{
    int z = x;
    Int sum = 1;
    while( z <= y ){
        sum = sum*z;
        z++
    }
}
```
Logic errors

- The computer will do precisely what you say even though it may not be what you want.
- What is wrong with this code?

```c
int sumBetween( int x, int y )
{
    int z = x;
    int sum = 1;
    while( z <= y )
    {
        sum = sum*z;
        z++;
    }
}
```