Introduction to computer programming A
LAB6

贾艳红 Jana
Email:jiayh@mail.sustech.edu.cn

LAB OBJECTIVES

1. Learn how to use static method
2. Learn how to use static method from other class
3. Learn how to use method overloading
4. Learn how to use two dimensional arrays
5. Learn to develop and invoke methods with array arguments and return values

knowledge points

1.1 Java method

1.2 Passing Parameters

1.3 Return value

1.4 Method Overloading

1.1 Java method

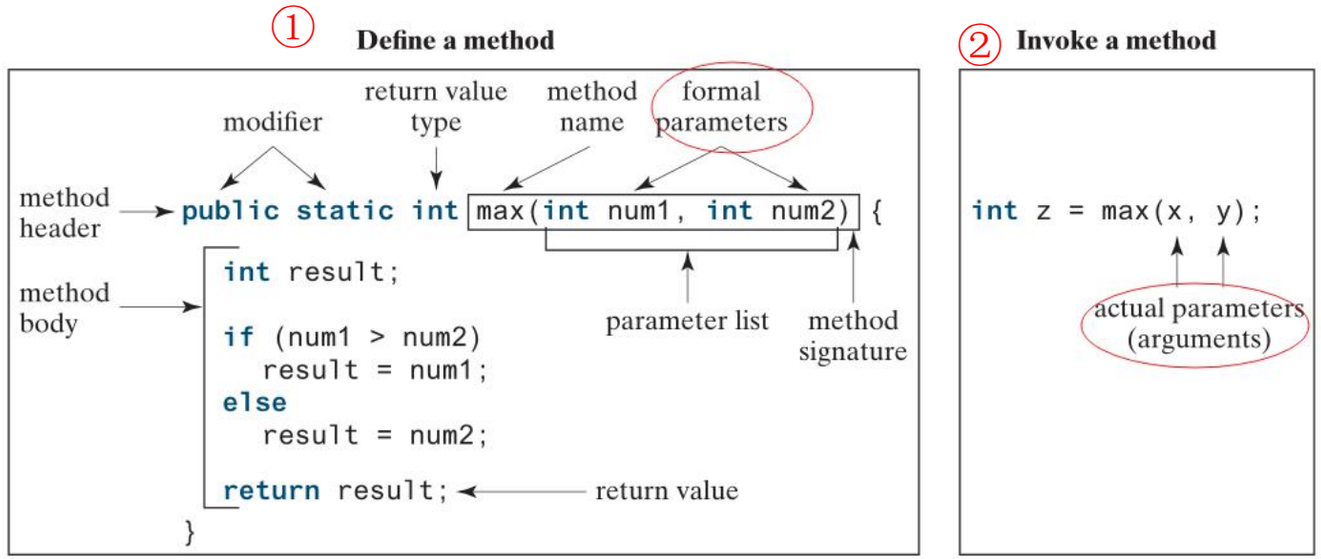
Methods can be of two broad categories. These are:

- **Standard Library Methods(such as print()、sqrt())**
- **User-defined Methods**

The syntax of User-defined Methods:

```
modifier returnValueType methodName(list of parameters) {
  // Method body;
}
```

the syntax for defining a method



A method can have zero or more parameters.

Example:

```java
class Main {

    public static void main(String[] args) {
        System.out.println("About to encounter a method.");

        // method call
        myMethod();

        System.out.println("Method was executed successfully!");
    }

    // method definition
    private static void myMethod(){
        System.out.println("Printing from inside myMethod()!");
    }
}
```

zero parameters

```java
public class TestMax {
    /** Main method */
    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);
    }

    /** Return the max of two numbers */
    public static int max(int num1, int num2) {
        int result;

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

        return result;
    }
}
```

i and j are actual parameters

num1 and num2 are formal parameters

1.2 Passing Parameters

Parameters can be passed in two ways: by value or by reference

By Value:

```java

public class TestPassByValue {
    /** Main method */
    public static void main(String[] args) {
        // Declare and initialize variables
        int num1 = 1;
        int num2 = 2;
        System.out.println("Before invoking the swap method, num1 is " + num1 + " and num2 is " + num2);
        // Invoke the swap method to attempt to swap two variables
        swap(num1, num2);
        System.out.println("After invoking the swap method, num1 is " + num1 + " and num2 is " + num2);
    }

    /** Swap two variables */
    public static void swap(int n1, int n2) {
        System.out.println("\tInside the swap method");
        System.out.println("\t\tBefore swapping, n1 is " + n1 + " and n2 is " + n2);
        // Swap n1 with n2
        int temp = n1;
        n1 = n2;
        n2 = temp;
        System.out.println("\t\tAfter swapping, n1 is " + n1 + " and n2 is " + n2);
    }
}
```

Output:

```
Before invoking the swap method, num1 is 1 and num2 is 2
        Inside the swap method
                Before swapping, n1 is 1 and n2 is 2
                After swapping, n1 is 2 and n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```

1.2 Passing Parameters

Parameters can be passed in two ways: by value or by reference

By reference:

```java
public class TestPassArray {
    /** Main method */
    public static void main(String[] args) {
        int[] a = { 1, 2 };
        // Swap elements using the swap method
        System.out.println("Before invoking swap");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
        swap(a[0], a[1]);
        System.out.println("After invoking swap");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");

        // Swap elements using the swapFirstTwoInArray method
        System.out.println("Before invoking swapFirstTwoInArray");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
        swapFirstTwoInArray(a);
        System.out.println("After invoking swapFirstTwoInArray");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    }
    /** Swap two variables */
    public static void swap(int n1, int n2) {
        int temp = n1;
        n1 = n2;
        n2 = temp;
    }
    /** Swap the first two elements in the array */
    public static void swapFirstTwoInArray(int[] array) {
        int temp = array[0];
        array[0] = array[1];
        array[1] = temp;
    }
}
```

Output:

```
Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}
```

1.3 Return value

```java
public class SquareMain {

    public static void main(String[] args) {
        int result, n;

        n = 3;                    3
        result = square(n);
        System.out.println("Square of 3 is: " + result);

        n = 4;
        result = square(n);
        System.out.println("Square of 4 is: " + result);
    }
    static int square(int i) {
        return i * i;             3
                  9
    }
}
```

```java

public class ExampleVoid {

    public static void main(String[] args) {
        methodRankPoints(255.7);
    }

The void keyword allows us to create methods which do not return a value.

    public static void methodRankPoints(double points) {
        if (points >= 202.5) {
            System.out.println("Rank:A1");
        }else if (points >= 122.4) {
            System.out.println("Rank:A2");
        }else {
            System.out.println("Rank:A3");
        }
    }
}

```

When you run the program, the output will be:

```
Squared value of 3 is: 9
Squared value of 4 is: 16
```

Output

```
Rank:A1
```

1.3 Return value

```java
    public static boolean[] getAbsent(int[][] records) {
        int sNum = records.length;
        int labNum = records[0].length;
        boolean[] abt = new boolean[sNum];
        // please complete your code here, must use
        for (int i = 0; i < sNum; i++) {
            int absentNum = 0;
            for (int j = 0; j < labNum; j++) {
                if (records[i][j] == 0)
                    absentNum++;
                if (absentNum > 1) {
                    abt[i] = true;
                    break;
                }
            }

        }

        return abt;
    }
```

return references

1.4 Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

There are three ways to overload the method in java:

1. **Number of arguments**

```
add(int, int)
add(int, int, int)
```

2. **Data type of parameters.**

```
add(int, int)
add(int, float)
```

3. **Sequence of Data type of parameters.**

```
add(int, float)
add(float, int)
```

Note: This is not a valid method overloading example. This will throw compilation error.

```
int add(int, int)
float add(int, int)
```

1.4 Method Overloading

Example: Changing no. of arguments

```java
//Method Overloading: changing no. of arguments
public class OverloadingDemo1 {

    public static void main(String[] args) {
        System.out.println(add(11, 11));
        System.out.println(add(11, 11, 11));

    }

    static int add(int a, int b) {
        return a + b;
    }

    static int add(int a, int b, int c) {
        return a + b + c;
    }

}
```

```
22
33
```

1.4 Method Overloading

Example: Changing data type of arguments

```java
//Method Overloading: changing data type of arguments
public class OverloadingDemo2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println(add(11,11));
        System.out.println(add(12.3,12.6));
    }

    static int add(int a, int b){return a+b;}
    static double add(double a, double b){return a+b;}

}
```

```
22
24.9
```

1.4 Method Overloading

Example: Changing Sequence of data type of arguments

```java
//Overloading – Sequence of data type of arguments
public class OverloadingDemo5 {

    public static void main(String[] args) {
        add(11, 11.0);
        add(12.3, 12);
    }

    static void add(double a, int b) {
        System.out.println(a+b);
        }

    static void  add(int a, double b) {
        System.out.println(a+b);
    }
}
```

```
22.0
24.3
```

1.4 Method Overloading

Why Method Overloading is not possible by changing the return type of method only?

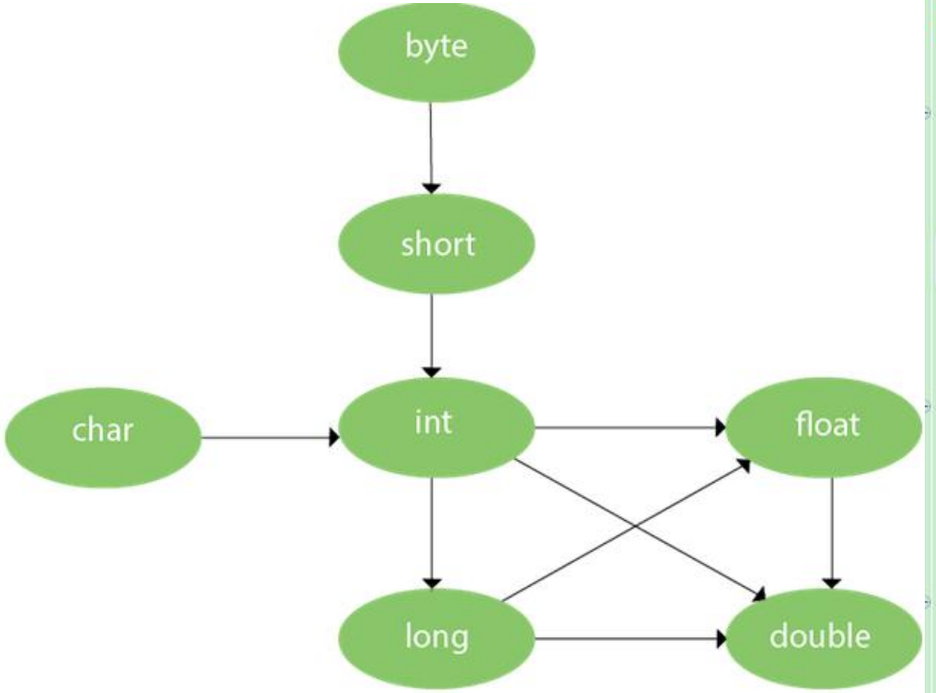
```java
//Why Method Overloading is not possible
//by changing the return type of method only?
public class OverloadingDemo3 {

    public static void main(String[] args) {
        System.out.println(add(11,11));//ambiguity

    }

    static int add(int a, int b) {
        return a + b;
    }

    static double add(int a, int b) {
        return a + b;
    }
}
```

```
G:\javaB_2019fall\LAB\LAB7\code_example>javac OverloadingDemo3.java
OverloadingDemo3.java:15: 错误: 已在类 OverloadingDemo3中定义了方法 add(int,int)
        static double add(int a, int b) {
```

1.4 Method Overloading

Type Promotion:One type is promoted to another implicitly if no matching datatype is found.

Example: Method Overloading with TypePromotion




```java
public class OverloadingDemo4 {

    public static void main(String[] args) {
        // now second int literal will be promoted to long
        sum(20, 20);
        sum(20, 20, 20);
    }

    static void sum(int a, long b) {
        System.out.println(a + b);
    }

    static void sum(int a, int b, int c) {
        System.out.println(a + b + c);
    }

}
```

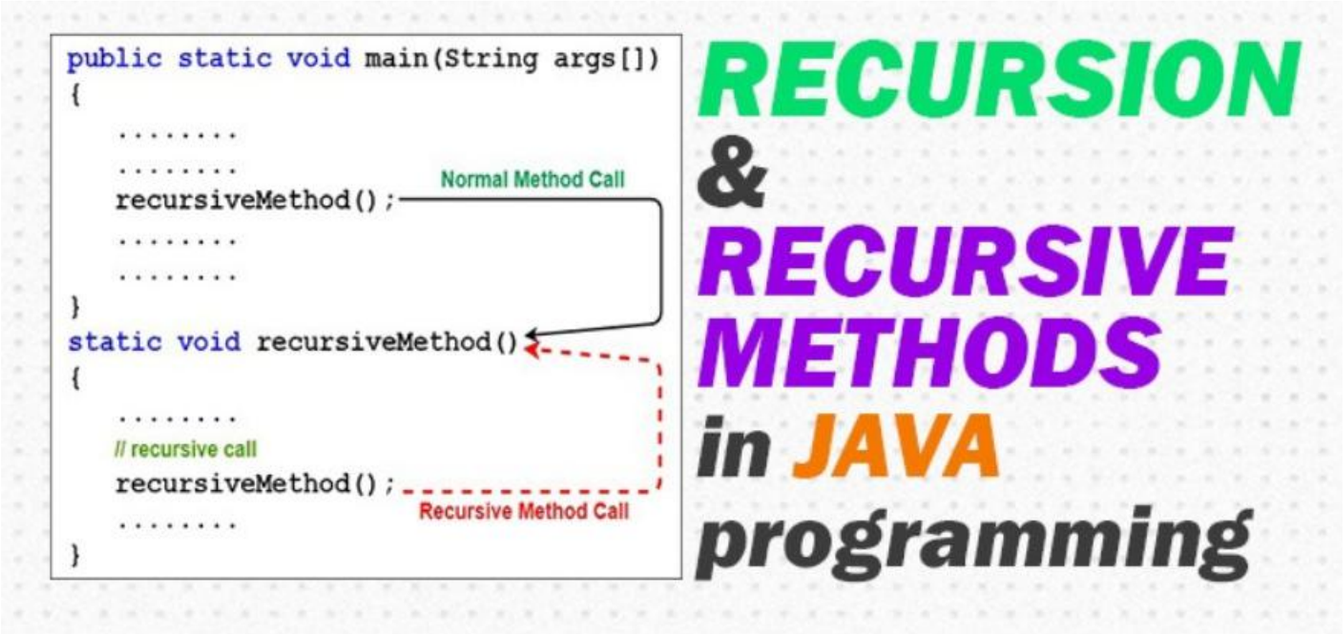
int

long

```
40
60
```

1.5 Recursive function

A function that calls itself is known as recursive function. And, this technique is known as recursion.



Recursion is used to solve various mathematical problems by dividing it into smaller problems.

1.5 Recursive function(case study)

Compute factorial of a number Factorial of n = 1*2*3...*n

```java
public class Factorial_1 {

    public static void main(String[] args) {
        int factorial =1;
        int n = 5;
        for (int i=1; i<=n; ++i)
            factorial *= i;
        System.out.println("Factorial of 5 is: "+factorial);
    }

}
```

output:

```
Factorial of 5 is: 120
```

Next,We can implement it with recursive functions!

1.5 Recursive function(case study)

Example : Compute factorial of a number Factorial of n = 1*2*3...*n

```java
public class Factorial {
    static int factorial(int n){
        if (n < 1)
            return 1;
        else
            return(n * factorial( n: n-1));
    }

    public static void main(String[] args) {
        System.out.println("Factorial of 5 is: "+factorial( n: 5));
    }
}
```

- Factorial function: f(n) = n*f(n-1),
- base condition: if n<=1 then f(n) = 1

4! = 3! * 4
= (2! * 3) * 4
= ((1! * 2) * 3) * 4
= (((0! * 1) * 2) * 3) * 4
= (((1 * 1) * 2) * 3) * 4 = 24

Base: 0! = 1

Recursion: n! = (n-1)! * n

return 5 * factorial(4) = 120
└ return 4 * factorial(3) = 24
└ return 3 * factorial(2) = 6
└ return 2 * factorial(1) = 2
└ return 1 * factorial(0) = 1

calling itself until the function reaches to the base condition!

1 * 2 * 3 * 4 * 5 = 120

Fig: Recursion

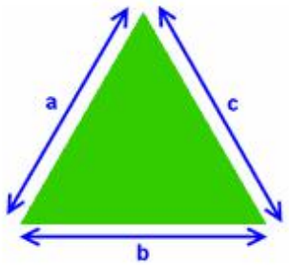
output:

```
Factorial of 5 is: 120
```

Exercises

Complete the exercises in the **2021S-Java-A-Lab-6.pdf** and submit to the blackboard as required.

Exercise 1



Area of a Triangle from Sides

You can calculate the area of a triangle if you know the lengths of all three sides, using a formula that has been known for nearly 2000 years.

It is called "Heron's Formula" after Hero of Alexandria (see below)

Just use this two step process:

Step 1: Calculate **"s"** (half of the triangles perimeter):

$$s = \frac{a+b+c}{2}$$

Step 2: Then calculate the **Area**:

$$A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

Example: What is the area of a triangle where every side is 5 long?

Step 1: $s = \frac{5+5+5}{2} = 7.5$

Step 2: $A = \sqrt{(7.5 \times 2.5 \times 2.5 \times 2.5)} = \sqrt{(117.1875)} =$ **10.825...**

Exercise 3

Fibonacci Sequence

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

- The 2 is found by adding the two numbers before it (1+1)
- The 3 is found by adding the two numbers before it (1+2),
- And the 5 is (2+3),
- and so on!

Example: the next number in the sequence above is 21+34 = **55**

THANK YOU

贾艳红 Jana
Email:jiayh@mail.sustech.edu.cn