Contents

1. Introduction
2. Types and Variables
3. Statements and Control Flow
4. Reading Input
5. Classes and Objects
6. Arrays
7. Methods
8. Scope and Lifetime
9. Utility classes
10. Introduction to Object-Oriented Analysis and Design
Chapter 7: Methods

After this chapter you will be able to:

- Define and call methods
- Return values from methods
- Understand call-by-value parameter passing
Methods

• A method is a procedure defined inside a class

• Methods:
  – facilitate abstraction
  – make programs modular
  – help with debugging by isolating problems
Method Definition

• Methods are defined inside classes
• void is the null type corresponding to no value

```java
public class Foo {
    public static void print1To3 () {
        for (int i = 1; i <= 3; i++)
            System.out.println(i);
    }
}
```
Main method

• main is a special method where execution starts

```java
class Foo {
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```
Calling Methods

• Syntax: name(val1,val2,...,valn)

```java
class Foo {
    public static void main (String [] args) {
        print1To3();
    }
    public static void print1To3 () {
        for (int i = 1; i <= 3; i++)
            System.out.println(i);
    }
}
```

Output

```
1
2
3
```
Calling Multiple Times

class Foo {
    public static void main (String [] args) {
        print1To3();
        print1To3();
    }
    public static void print1To3 () {
        for (int i = 1; i <= 3; i++)
            System.out.println(i);
    }
}

Output

1
2
3
1
2
3
Return

• A method can *return* a value

```java
public static int sum (int x, int y) {
    return x + y;
}
```
Returned Value Must Match

• A method can only return values of the type specified by the return type

```java
public static int sum (int x, int y) {
    return "hi there";
}
```

• The compiler will reject this program. Why?
Calling `sum`

class Foo {
    public static void main (String [] args) {
        int result = sum(5,3);

        System.out.println("the result is: " + result);
    }
    public static int sum (int x, int y) {
        return x + y;
    }
}

• Question: What is the output?
Placing Calls Inside Expressions

- Can insert a call to \textit{sum} anywhere an integer value is required

```java
System.out.println("the result is: " + sum(10,20));

int a = 5 * sum(6,12);

int b = 22 * sum(5,3) - sum(12,33);
```
Calling *sum* Multiple Times

- Can call *sum* multiple times yielding different results

```java
class Foo {
    public static void main (String [] args) {
        System.out.println("4 + 9 = " + sum(4,9));
        System.out.println("12 + 5 = " + sum(12,5));
    }
    public static int sum (int x, int y) {
        return x + y;
    }
}
```

Output

```
4 + 9 = 13
12 + 5 = 17
```
Passing Primitive Variables

• In Java parameters are passed by value

• By value means that copies of argument variables are passed to a method

```java
public static void assign (int x) {
    x = 2;
}
```

```java
int a = 4;
assign(a);
System.out.println("a = " + a);
```

Output

```
a = 4
```

```
a = 4
```
Passing Object Variables

• Same as before but now we make copies of addresses instead of values

```java
public static void assign (String x) {
    x = "java";
}

String a = "rules";
assign(a);
System.out.println("a = " + a);
```

Output

```
a = rules
```
Visible Side Effects

- This time the effect of the assignment propagates

```java
public static void assign (int [] x) {
    x[0] = 4; // modifies object, not // address
}
```

```java
int [] a = {5,6,7};
assign(a);
System.out.println("a[0] = "+ a[0]);
```

Output  a[0] = 4
Parameter Passing Summary

• The effect of assignments to parameters of **primitive type** are **not visible** to the caller

• The effect of assignments to parameters of **object type** are **not visible** to the caller

• If the **object** that a parameter points to is modified then this effect **is visible** to the caller
Power

```java
public static int power (int base, int exp) {
    int result = 1;
    for (int i = 1; i <= exp; i++)
        result *= base;
    return result;
}
```

```java
System.out.println("2 to the 3rd = " + power(2,3));
```

Output

2 to the 3rd = 8
Reading a Line

```java
public static String getLine() {
    int b;
    String result = "";
    try {
        while ((b = System.in.read()) != '\n')
            result += (char) b;
    } catch (java.io.IOException e) {}
    return result;
}
```
Revised Echo

```java
public class Echo {
    public static void main (String [] args) {
        int b;
        while (true) {
            System.out.println("type in a line to echo");
            System.out.println(getLine());
        }
    }

    public static String getLine() {
        int b;
        String result = "";
        try {
            while ((b = System.in.read()) != '\n')
                result += (char) b;
        } catch (java.io.IOException e) {}
        return result;
    }
}
```
Execution

• As before, user’s lines are echoed

Input/Output

<table>
<thead>
<tr>
<th>type in a line to echo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ok, how about this ?</td>
</tr>
<tr>
<td>ok, how about this ?</td>
</tr>
<tr>
<td>type in a line to echo</td>
</tr>
<tr>
<td>here is another</td>
</tr>
<tr>
<td>here is another</td>
</tr>
</tbody>
</table>
Recursion

• A method may call itself recursively

• Recursion facilitates problem decomposition by allowing a problem to be broken down into identical sub problems

• Example: The sum of the numbers from 1 to n equals n plus the sum of the numbers from 1 to n-1
Recursive Sum

```java
public static int sumToN (int n) {
    if (n == 1)
        return 1;
    else
        return n + sumToN(n-1);
}
```

```java
System.out.println("sumToN(5) = " + sumToN(5));
```

Output: `sumToN(5) = ?`
It’s Exercise Time