CIS 351-Data Structure-Methods
Jan 23, 2020

Dr. Farzana Rahman

Syracuse University

Advantage of Methods

- **Avoid code repetition** – Methods are re-usable. We can mix and match existing methods to solve new problems.
- **Simplify problem-solving** – Humans can only solve large problems by decomposing them into smaller problems, which may themselves need to be broken into smaller problems...
- **Simplify testing** – Individual methods may be tested in isolation.

Terminology

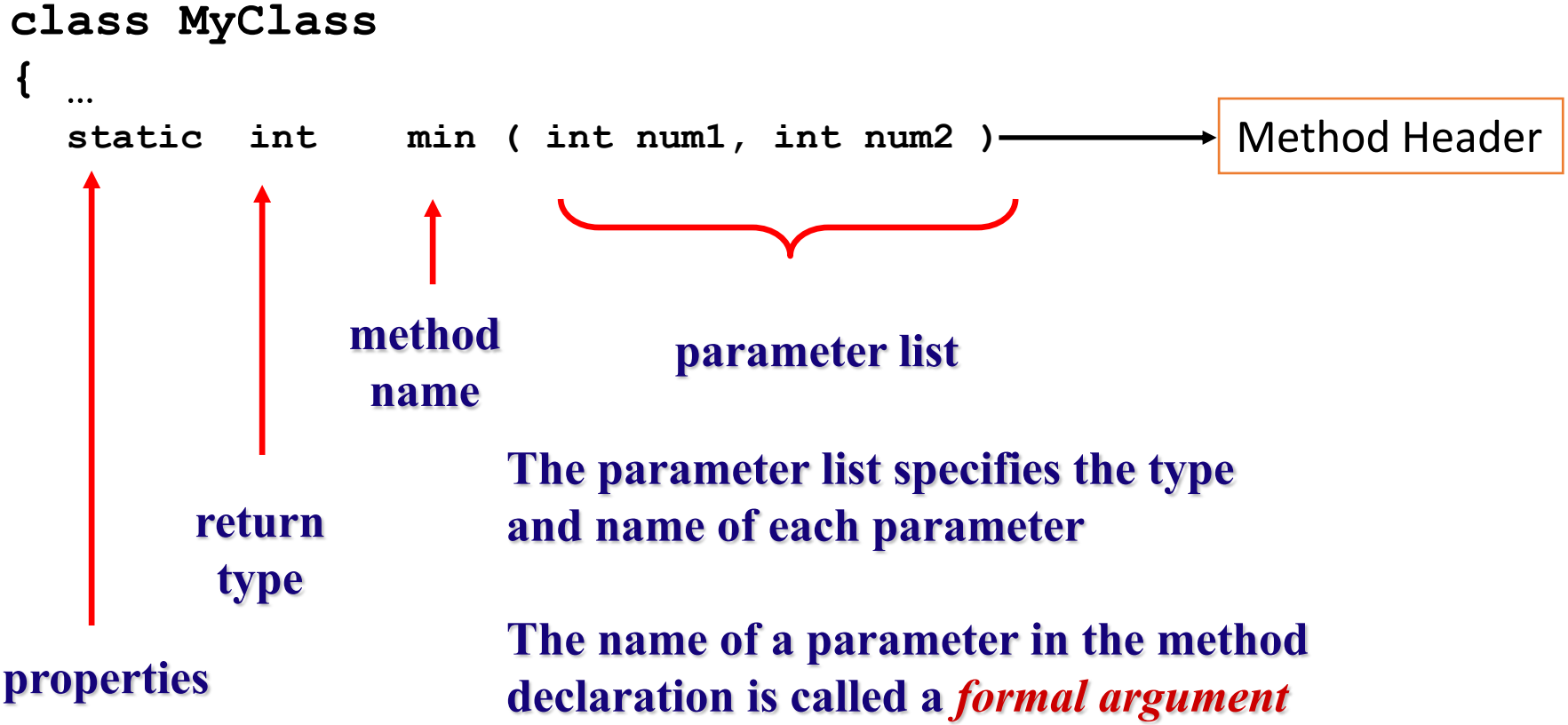
- There are several names for the same general idea:
  - procedure (most often used when no value is returned)
  - function (most often used when a value *is* returned)
  - Subroutine
  - method

Methods

- Methods we have seen so far:
  - main()
  - println()
  - nextInt(), nextDouble()… … …
- Methods are use to break a complex program into small, manageable pieces
- In Java, each method is defined within specific class
- Method takes input, performs actions, and produces output
  - Void methods terminates after completing work, ex. `main()` method
  - Value returning method returns a value, ex. `String.length()`

Method Declaration: Header

- A method declaration begins with a *method header*



Method Declaration: Body

The header is followed by the *method body:*

```
…
static int min(int num1, int num2)
{
    int minValue = num1 < num2 ? num1 : num2;
    return minValue;
}

…
```

The `return` Statement

- The *return type* of a method indicates the type of value that the method sends back to the calling location
  - Method that does not return a value has a `void` return type

- The *return statement* specifies the value that will be returned
  - Its expression must conform to the return type

Flow of Execution

- Execution always begins at the first statement in the function `main`
- Other functions are executed only when called
- Function prototypes appear before any function definition
  - Compiler translates these first
- Compiler can then correctly translate a function call

User-Defined Functions

- **Value-returning functions:** have a return type
  - Return a value of a specific data type using the `return` statement
- **Void functions:** do not have a return type
  - *Do not* use a `return` statement to return a value

```
void functionName(formal parameter list)
{
    statements
}
```

Scope of an Identifier

- Scope of an identifier: where in the program the identifier is accessible
- Local identifier: identifiers declared within a function (or block)
- Global identifier: identifiers declared outside of every function definition

Sample question – Method definition

Write a public method named calculateMax that takes two double variable parameter input and returns the maximum of the two

```
public double calculateMax(double num1, double num2)
{… …}
```

How do we call this method?

```
double max = calculateMax(455.5,100.00);
```

Or

```
double d1 = 455.5;
double d2 = 100.00;
double max = calculateMax(d1, d2);
```

The method parameters and the argument data types has to be exactly same

Two issues

- Scope – The region of code where a variable can be seen/accessed.
  - Variables defined inside methods are called local variables – visible only inside that method.

- Pass by value – In Java, methods receive a copy of their arguments Changing the parameter variable only changes the copy.

Quiz 2: What Will Be Printed?

```
 1 public class MethodDemo {
 2
 3    public static int methodOne(int a, int b) {
 4       int result;
 5
 6       result = a * 2 + b;
 7       return result;
 8    }
 9
10    public static void main(String[] args) {
11        System.out.println(methodTwo(4, 5));
12    }
13
14    public static String methodTwo(int a, int b) {
15       String result;
16
17       result = "answer: " + methodOne(b, a);
18       return result;
19    }
20 }
```

Answer = 14

Quiz 3: What Will Be Printed?

```
public class MethodDemo {

    public static int methodOne(int a, int b) {
        int result;

        result = a * 2 + b;
        return result;
    }

    public static void main(String[] args) {
        methodTwo(4, 5);
        methodOne(3, 4);
    }

    public static String methodTwo(int a, int b) {
        String result;

        result = "answer: " + methodOne(b, a);
        return result;
    }
}
```

Answer = 14

10