Lecture 05: Methods

AITI Nigeria Summer 2012
University of Lagos.

MISTI

Agenda

- What a method is
- Why we use methods
- How to declare a method
- The four parts of a method
- How to use (invoke) a method
- The purpose of the main method

The Concept of a Method

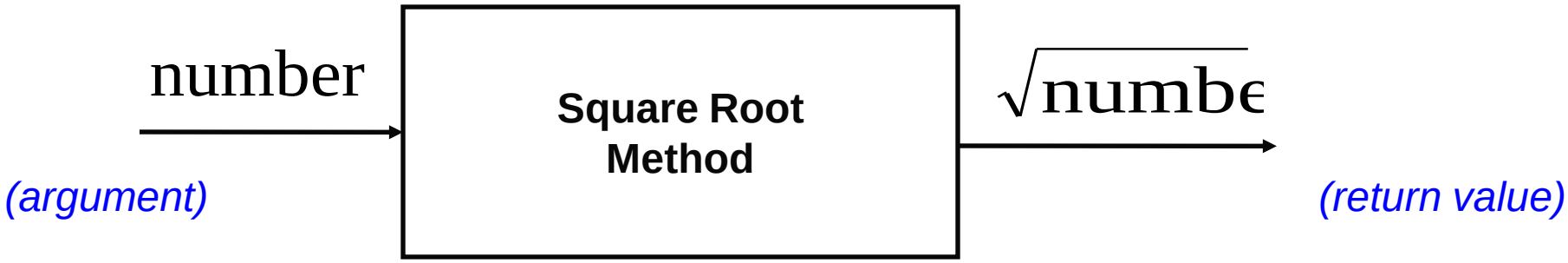
- Methods are a way of organizing a sequence of statements into a named unit.
  - Reusable
  - Parameterizable (can accept inputs)
  - Organize code into smaller units
    - Easier to understand
- Any complex process that can exist on its own should be a method
  - Better to have more methods, even if they are not reused.

The Concept of a Method

- Methods can accept inputs (called arguments)
- They can then perform some operations with the arguments
- And can output a value (called a return value) that is the result of the computations

Square Root Method

- The square root method accepts a single number as an argument and returns the square root of that number.

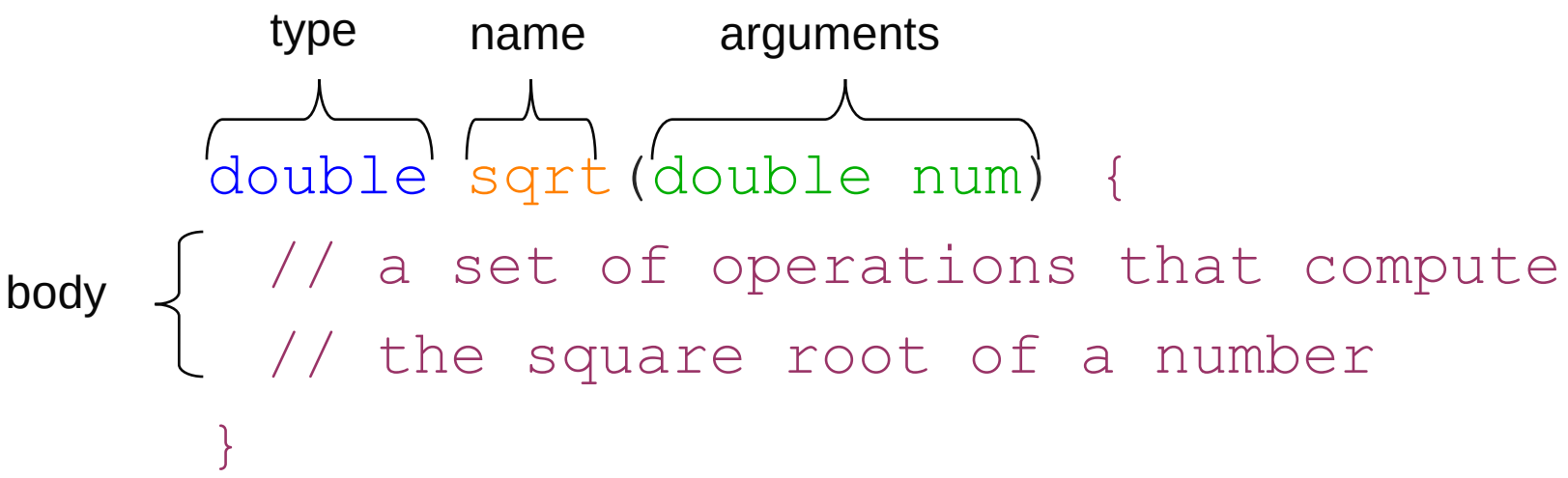


Square Root Method (con't)

- The computation of square roots involves many intermediate steps between input and output.
- When we use square root, we don't care about these steps or details. All we need is to get the correct output.
- Hiding the internal workings of a method and providing the correct answer is known as *abstraction*

Declaring Methods

- A method has 4 parts: the return type, the name, the arguments, and the body:



- The type, name and arguments together is referred to as the *signature* of the method
- Methods with same names must have unique signature

Return Type of a Method

- The return type of a method may be any data type....

- The return type of a method is a promise for what data type the output will be
  - A method can return different outputs than inputs
  - A method cannot return multiple types, returns one type
- Methods can also return nothing in which case they are declared void.

Return Statements

- The return statement is used in a method to output the result of the method computation.
- It has the form:
  - `return expression-value;`
- The type of the expression_value must be the same as the type of the method:

```
double sqrt(double num) {
    double answer;
    // Compute the square root of num
    // and store in answer
    return answer;
}
```

- What is the return type of this method?

Return Statements

A method exits immediately after it executes the return statement

```
double sqrt(double num) {
  double answer;
  // Compute the square root of num
  // and store in answer
  return answer;

  answer = 5 + 4; //never executed, illegal
}
```

Multiple Returns

- An example using multiple returns:

```
int absoluteValue (int num) {
 if (num < 0)
    return –num;
 else
    return num;
}
```

void Methods

- A method of type **`void`** does not return a value

- Used often in practice.
  - Perform some computation that does not produce a value
  - Affect system state, ex: System.out.println()
- A void method can have a return statement without any specified value. i.e. **`return;`**
- If no return statement is used in a method of type void, it automatically returns at the end

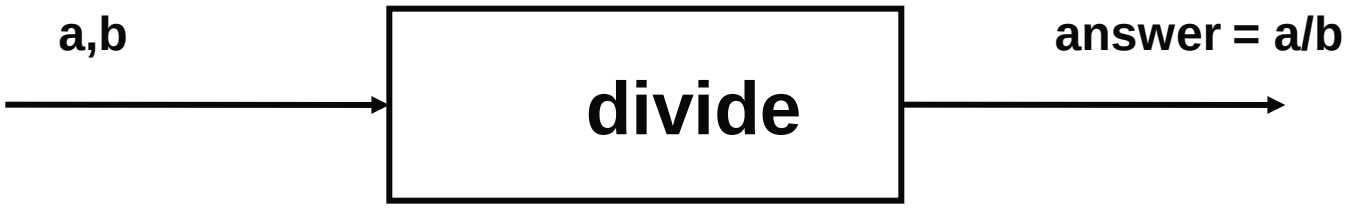
Method Arguments

- Methods can take input in the form of arguments.
- Arguments are used as variables inside the method body.
- Like variables, arguments must have their type specified.
- Arguments are specified inside the parentheses that follow the name of the method.

Example Method

- Here is an example of a method that divides two doubles:

```
double divide(double a, double b) {
  double answer;
  answer = a / b;
  return answer;
}
```



Method Arguments

- Multiple method arguments are separated by commas:

```
double divide(double a, double b) {
    double answer;
    answer = a / b;
    return answer;
}
```

- Arguments may be of different types (double/int)
  - `double divide(int a, int b)`
- When calling method, exact sequence of input types must be applied

The Method Body

- The body of a method is a block specified by curly brackets i.e { }. The body defines the actions of the method.
- The method arguments can be used anywhere inside of the body.
- All methods must have curly brackets to specify the body even if the body contains only one statement or no statements.

```
double divide(
 double a, double b)
 {
    double answer;
    answer = a / b;
    return answer;
 }
```

Invoking Methods

- To call a method, specify the name of the method followed by a list of comma separated arguments in parentheses:

```
divide(10, 2); //Computes 10/2
```

- If the method has no arguments, you still need to follow the method name with empty parentheses:

```
int size() {
  //Compute and return size
 }
…

size(); //Calls size
```

Method Variable Scoping

- For now, methods can only access their own arguments and local variables.
  - A method cannot access arguments/locals from other methods
  - Even if one method calls another

- Example…

Recursive Methods

- A method can also call itself!
  - When a method calls itself, it needs a stopping condition, called the base case
    - Or else it would call itself without end
  - Example Factorial:
- Factorial of n, denoted n!:
  - $n \times (n - 1) \times (n - 2) \times \ldots \times 0$
  - $0! = 1$ (base case)

Factorial Implementation

```
int factorial(int n) {

    if (n==0)
      return 1;
    else {
      return n *
            factorial (n-1);
    }
}
```

Static Methods

- For now, all the methods we write in lab will be static.

```
static double divide(double a,
                     double b) {
   return a / b;
}
```

- We'll learn what it means for a method to be static in a later lecture

main – A Special Method

- The only method that we have used in lab up until this point is the **`main`** method.
- The main method is where a Java program always starts when you run a class file (entry point)
- The **`main`** method is static and has a strict signature which must be followed:

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

main Method (con't)

```
class SayHi {
   public static void main(String[] args) {
     System.out.println("Hi, " + args[0]);
  }
}
```

- If you were to type `java Program arg1 arg2 … argN` on the command line, anything after the name of the class file is automatically entered into the `args` array:

```
java SayHi Sonia
```

- In this example `args[0]` will contain the String "Sonia", and the output of the program will be "Hi, Sonia".

Methods Review

- What are the four parts of a method and what are their functions?
    1. **Return type** – data type returned by the method
    2. **Name** – name of the method
    3. **Arguments** – inputs to the method
    4. **Body** – sequence of instructions executed by the method

What is wrong with the following?

```
static double addSometimes(num1, num2){
    double sum;
    if (num1 < num2){
     sum = num1 + num2;
     String completed = “completed”;
     return completed;
    }
}
```

- Types for the arguments num1 and num2 are not specified
- String completed does not match the correct double return type
- Method addSometimes does not always return an answer. This will cause an error in Java because we specified that addSometimes would always return a double.

Example

```
class Max {
    public static void main(String args[]) {
        if (args.length == 0) return;

        int max = Integer.parseInt(args[0]);
        for (int i=1; i < args.length; i++) {
            if (Integer.parseInt(args[i]) > max) {
                max = Integer.parseInt(args[i]);
     }
     }
     System.out.println(max);
     }
}
```

After compiling, if you type `java Max 3 2 9 2 4`
the program will print out `9`

Important Points Covered ….

- Methods capture a piece of computation we wish to perform repeatedly into a single abstraction
- Methods in Java have 4 parts: return type, name, arguments, body.
- The return type and arguments may be either primitive data types (i.e. int) or complex data types (i.e. Objects), which we will cover next lecture
- **main** is a special Java method which the java interpreter looks for when you try to run a class file
- **main** has a strict signature that must be followed:

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