Session 05: Methods

Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

Lecture outline


Introducing Methods


Passing Parameters


Overloading Methods


Scope of Local Variables


Method Abstraction


The Math Class


Case Studies

Why do we need methods?

- Suppose that you need to find the sum of integers from 1 to 10 , from 20 to 30 , and from 35
- to 45 , respectively. You may write the code as follows:

```
int sum = 0;
for (int i = 1; i <= 10; i++){
        sum += i;
}
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++){
        sum += i;
}
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++){
        sum += i;
}
System.out.println("Sum from 35 to 45 is " + sum);
```

Methods

```
1       public static int sum(int i1, int i2) {
2       int sum = 0;
3       for (int i = i1; i <= i2; i++){
4               sum += i;
5
6       return sum;
7       }
8
9       public static void main(String[] args) {
10              System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11              System.out.println("Sum from 20 to 30 is " + sum(20, 30));
12              System.out.println("Sum from 35 to 45 is " + sum(35, 45));
13      }
```

Introducing Methods

A Method is a group of statements and variables that is given a name and may be called upon by this name from elsewhere.

Java API provides many pre-written methods e.g println ()

A large program can be hard to maintain, so divide and conquer

Split large sections of code into self contained blocks
Declare variables inside a method for local use

Data can be passed into and out of Methods e.g. println(“Hi”)

println() method can display any text (a parameter)

Methods promote software re-use

Methods help you keep track of complex programs

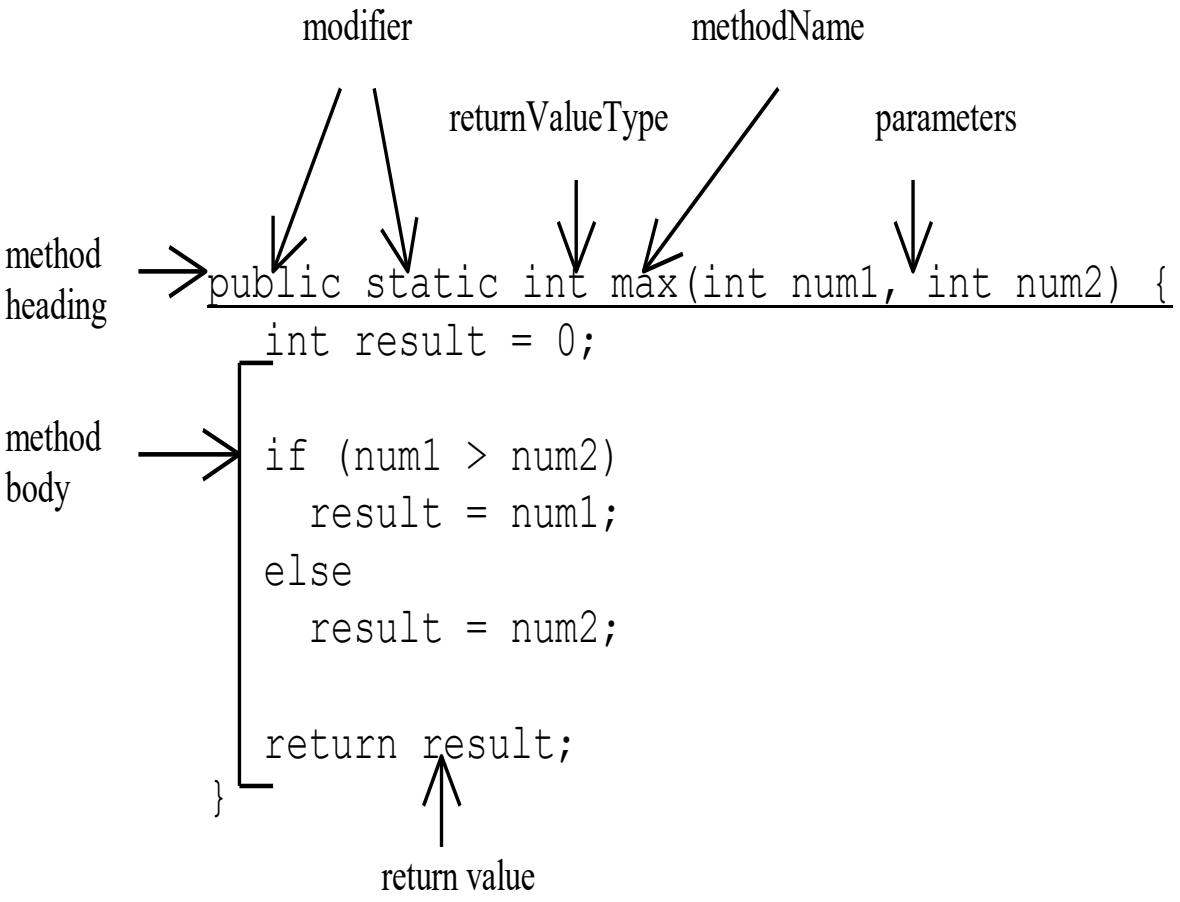
Methods are always part of a class

A Class is merely a collection of Methods and Variables

Methods

A method is a collection of statements that are grouped together to perform an operation.

Method Structure



Introducing Methods, cont.

- *parameter profile* refers to the type, order, and number of the parameters of a method.
- *method signature* is the combination of the method name and the parameter profiles.
- The parameters defined in the method header are known as *formal parameters*.
- When a method is invoked, its formal parameters are replaced by variables or data, which are referred to as *actual parameters*.
- Methods can return values to their caller – the Method declaration includes the return value type – can be void

Declaring Methods

```
public static int max (int num1, int num2) {
if (num1 > num2)
    return num1;
  else
    return num2;
}
```

Methods have a type associated with them. The declaration above expects 2 ‘int’ parameters, and returns a result of type ‘int’.

This declaration expects no parameters, and returns nothing :

```
public static void test (void) {
 ....
}
```

Calling a Method

```
int i = 5; int j = 2;       // Initialise parameters
int k = max(i, j);  // Call the max method & save
result
```

Testing the `max` method

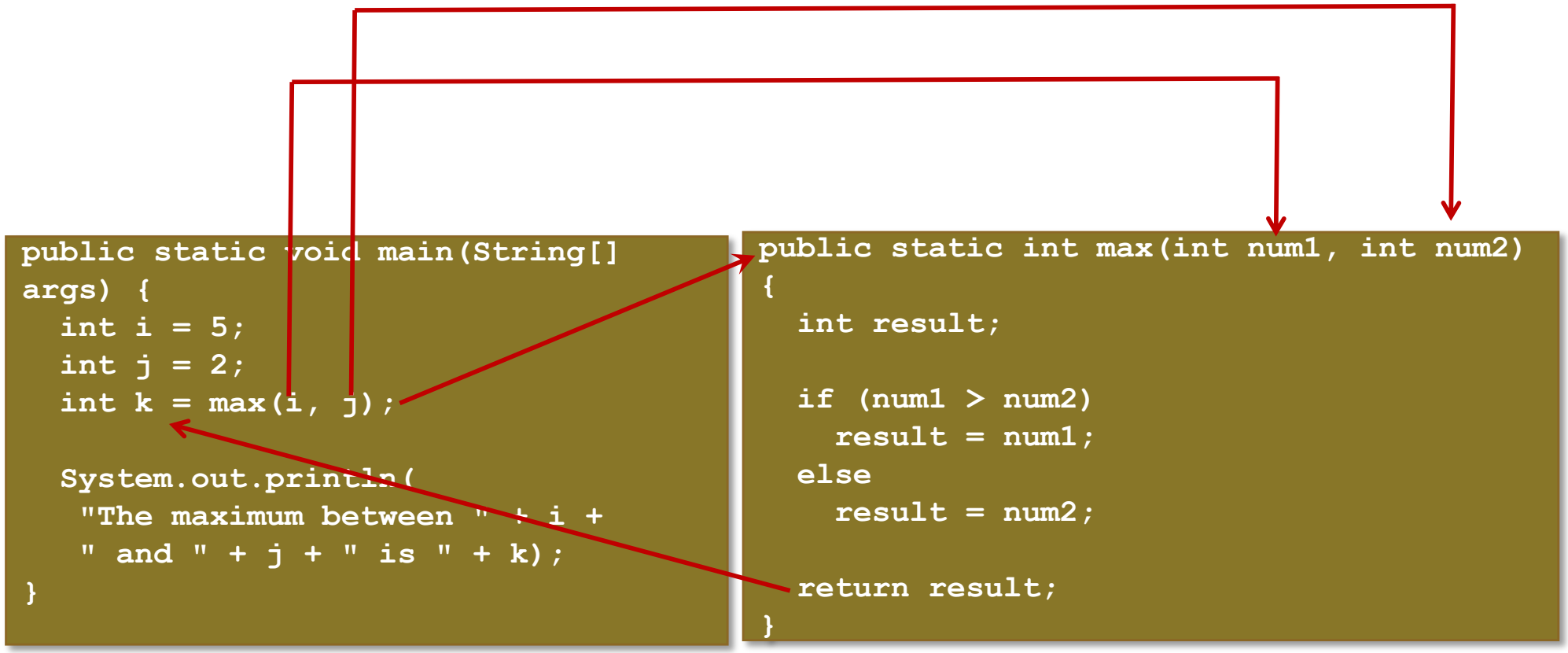
This program demonstrates calling a method max to return the largest of the `int` values

*(in Netbeans)

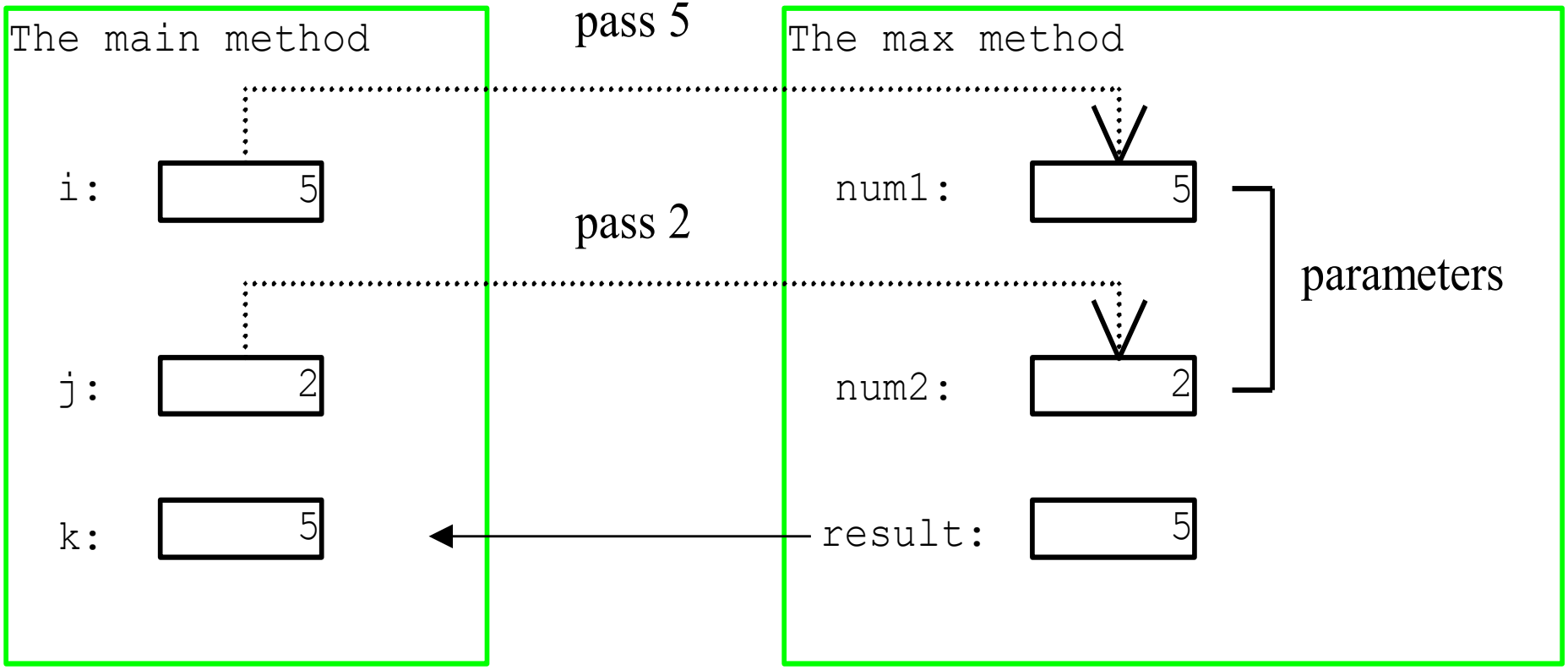
TestMax

Run

Calling Methods, cont.



Calling Methods, cont.



CAUTION

A return statement is required for a non-void method.

The following method is logically correct, but it has a compilation error, because the Java compiler thinks it possible that this method does not return any value.

```
public static int xMethod(int n) {
  if (n>0) return 1;
  else if (n==0) return 0;
  else if (n<0) return -1;
}
```

To fix this problem, delete if (n<0) in the code.

Passing Parameters

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

- Any Java type can be used as a parameter, however, a call to a Method must agree in Type, Order and Number of parameters.
- Java passes parameters of simple types (char, byte, int, float etc) by value, i.e. a copy of the parameter is made and passed to the method, the original stays the same.
- In the example above, **int n** is a simple type, **String message** is a reference to a more complex object.

Pass by Value

Testing Pass by value

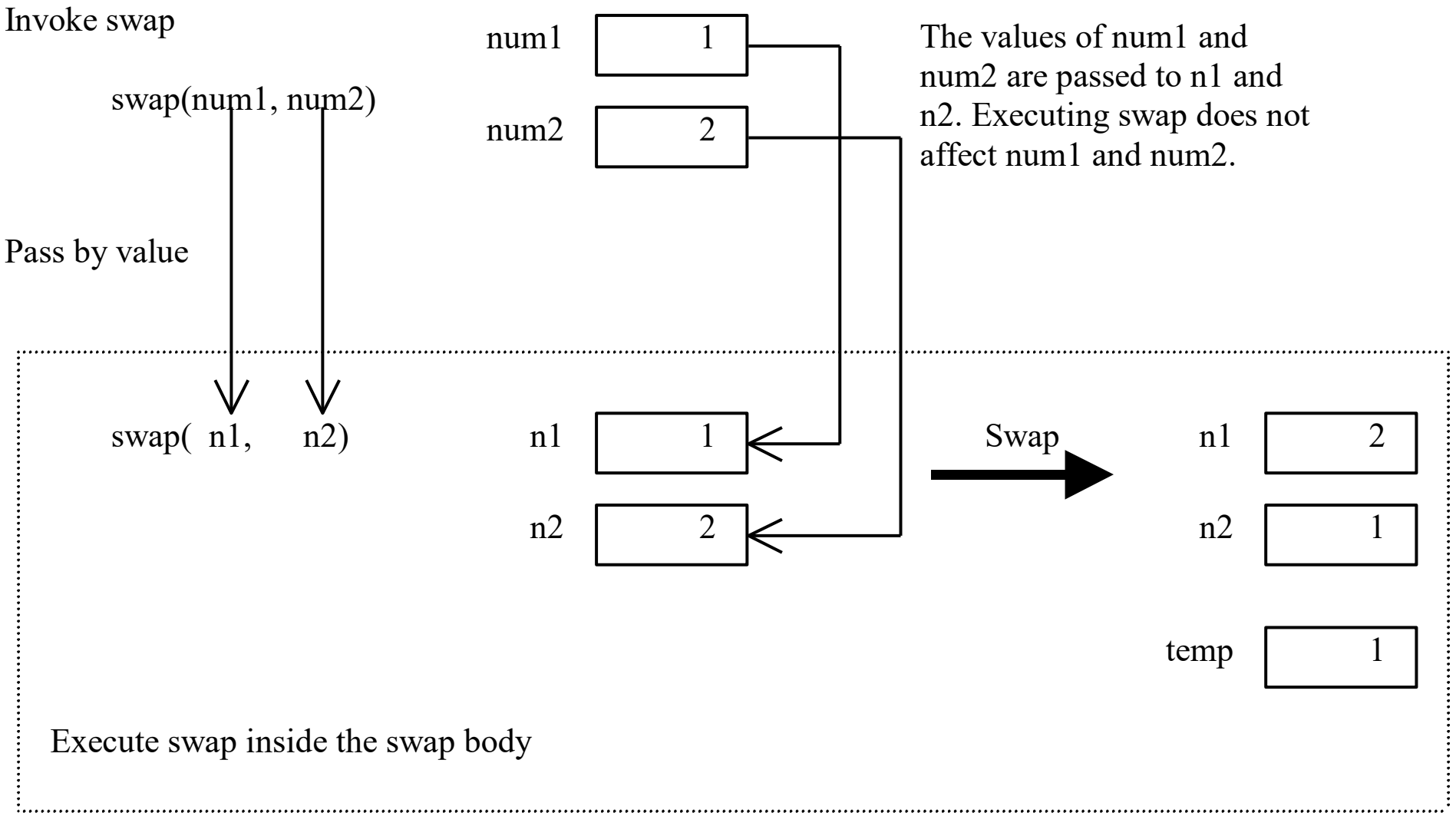
This program demonstrates passing values to the methods.

(Netbeans testPassByValue.java)

TestPassByValue

Run

Pass by Value, cont.



Scope of Local Variables

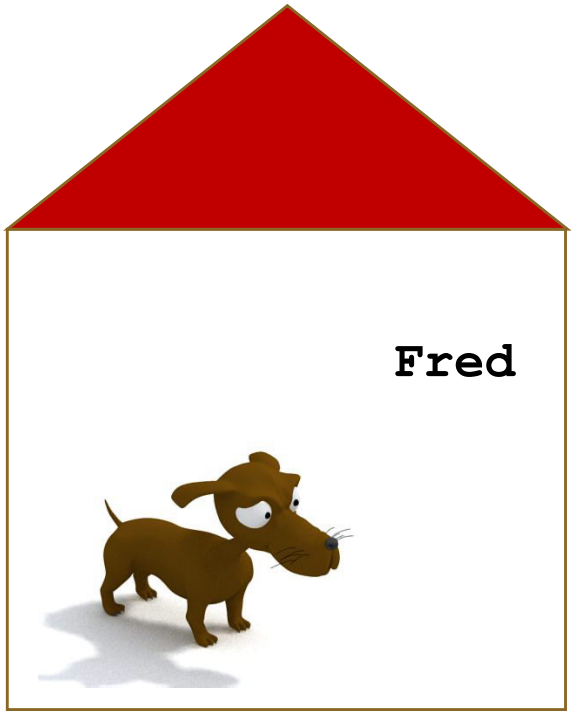
- A local variable: a variable defined inside a method.
- Scope: the part of the program where the variable can be referenced.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.
- Variable declared outside a Method are ‘Global’ – re Classes
- You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks. Thus, the following code is correct.

Scope of Local Variables


```
public void myHouse(){

Dog Fred = new Dog();

}
```

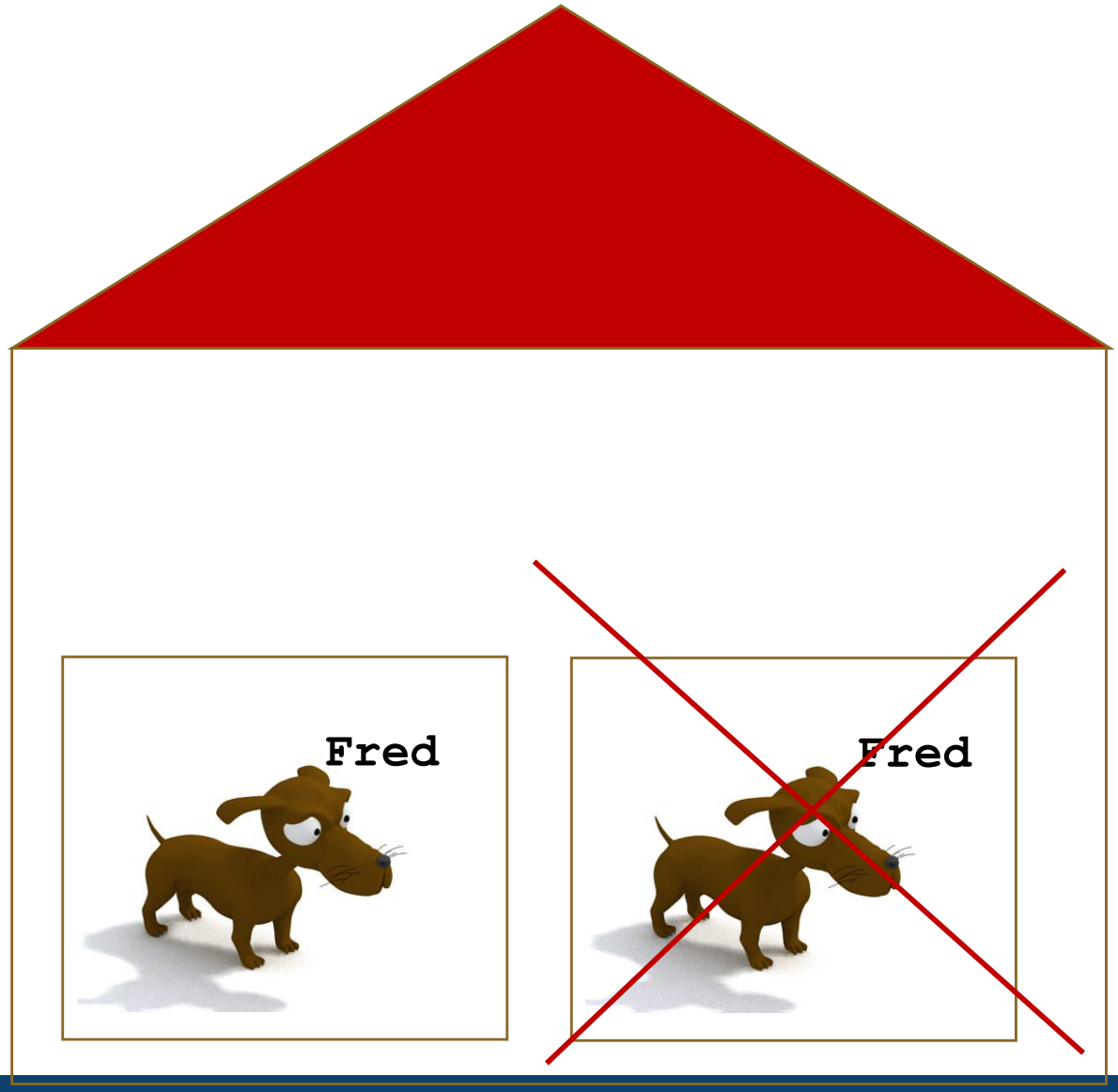



```
public void otherHouse(){

Dog Fred = new Dog();

}
```

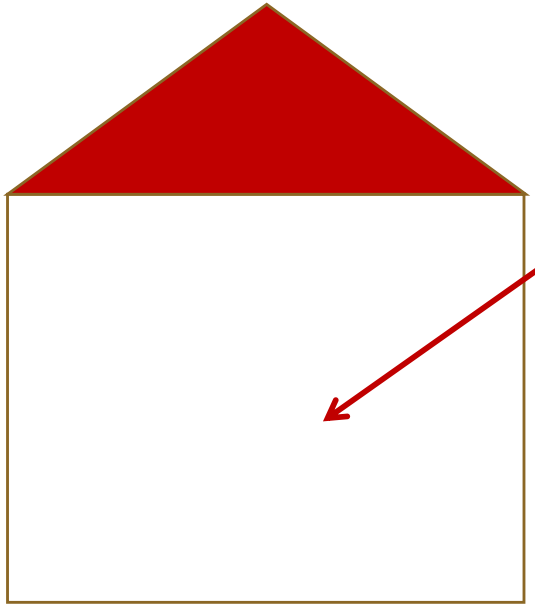
Scope of Local Variables (cont.)

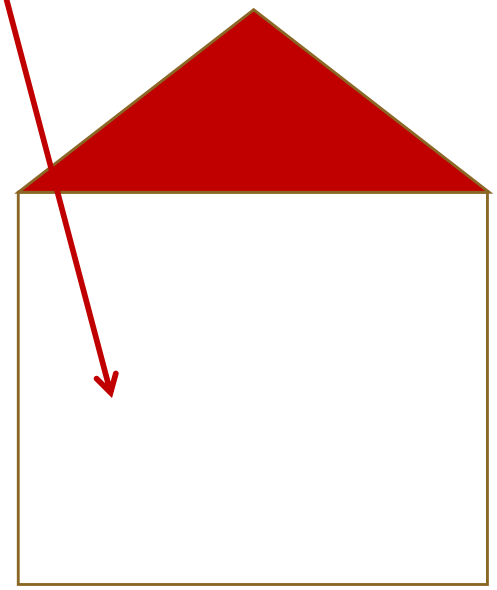


Global Variable

Fred

```
Dog Fred = new Dog;
```





```
public void myHouse(){

Fred.bark();
}
```

```
public void otherHouse(){

Dog Fred = new Dog();

}
```

Scope of Local Variables, cont.

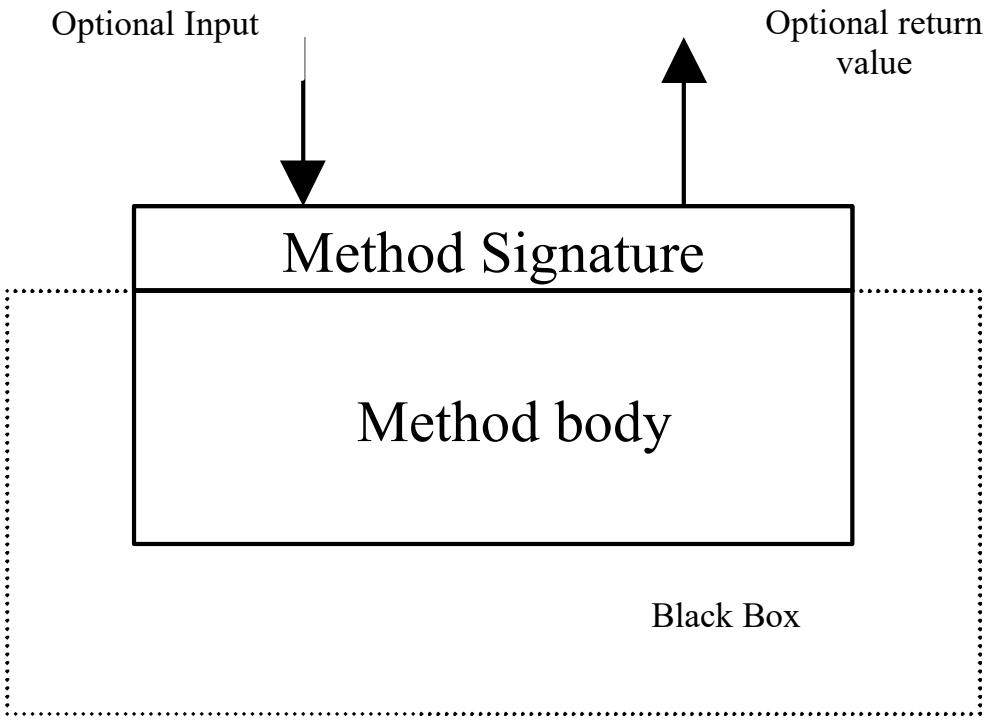
```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Scope of Local Variables, cont.

```
// Generates compiler error
// (x already defined)
//
int x = 2;
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;                    // declared already
    x += i;
  }
}
```

Method Abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.



Benefits of Methods

- Write once and reuse may times.
- Information Hiding – The internal processing and data structures should be private and hidden from the rest of program
- Reduce complexity – add structure to a program.
- Modest sized – small enough to keep within ones intellectual span of control – eg one printed page of code
- Method should be self contained, and its removal from a program should only disable its unique feature
- Software maintenance – important to minimise the interactions between modules

Why use Methods ?

- For simple and short lived programs, a quick ‘hack’ might suffice – no structure, no methods.
- Software is very expensive to write, test and debug. Trend is towards re-using software where ever possible.
- Most Java facilities are implemented as Methods within Class libraries – pre written by others
- Modularising software aims to create independent units with high Cohesion (single function) and low Coupling (interaction between modules)
- Good modularity greatly reduces maintenance effort.
- Commercial software – up to 60% of total lifecycle cost of producing software is spent on maintenance after release.

Case Studies

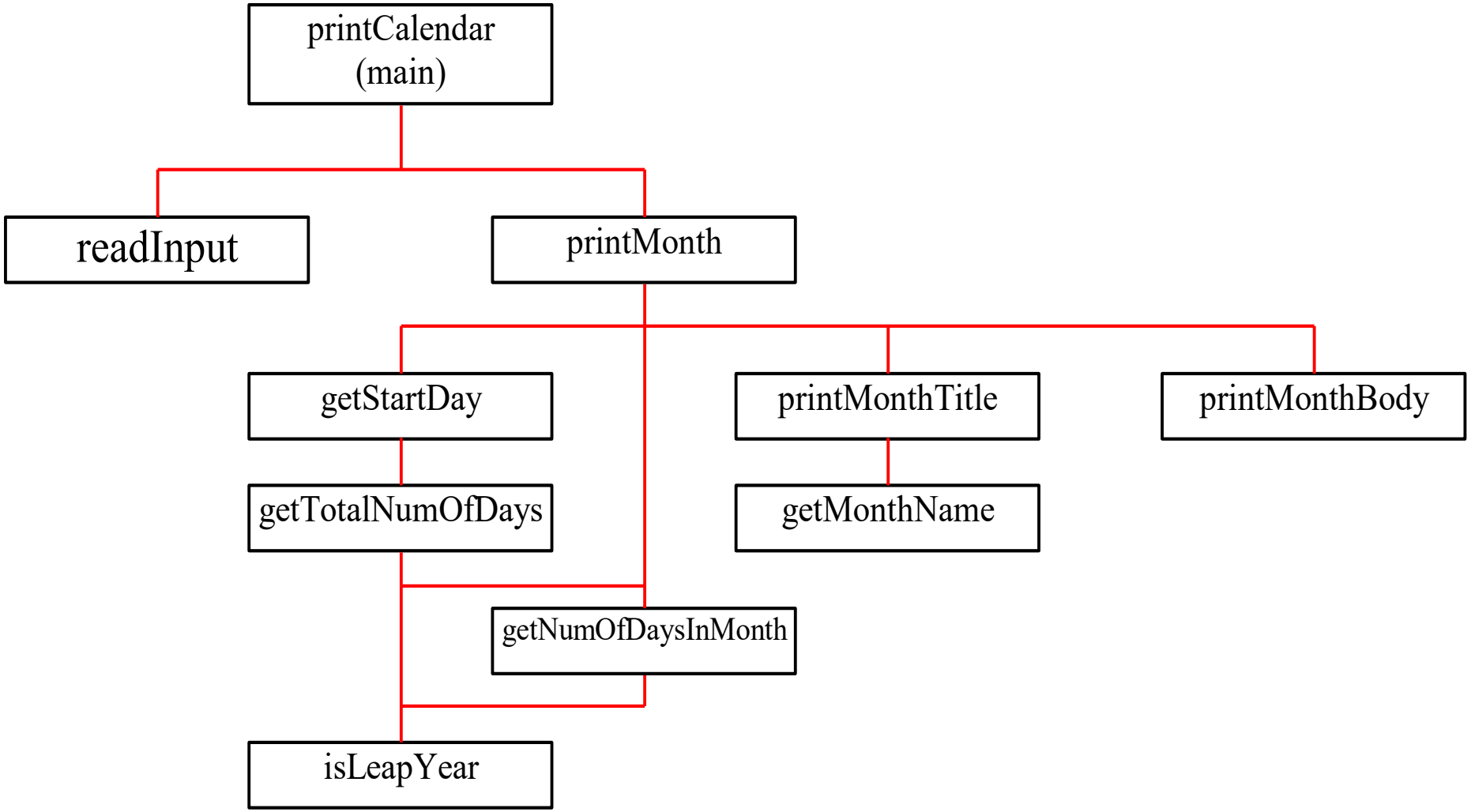
Displaying Calendars

The program reads in the month and year and displays the calendar for a given month of that year.

PrintCalendar

Run

Design Diagram



The `Math` Class

- use *import java.math.*;*
- Class constants:
  - `PI`
  - `E`
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Exponent Methods

- **`exp(double a)`**
  Returns `e` raised to the power of `a`.
- **`log(double a)`**
  Returns the natural logarithm of `a`.
- **`pow(double a, double b)`**
  Returns `a` raised to the power of `b`.
- **`sqrt(double a)`**
  Returns the square root of `a`.

Rounding Methods

- ***Math.ceil(double x)***

  x rounded up to its nearest integer. This integer is returned as a double value.

  e.g.

```
>Math.ceil(2.3); // = 3.0
>Math.ceil(2.9); // = 3.0
```

- ***double floor(double x)***

  x is rounded down to its nearest integer. This integer is returned as a double value.

  e.g.

```
>Math.floor(2.3); // = 2.0
>Math.floor(2.9); // = 2.0
```

Rounding Methods

- ***double rint(double x)***

  x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.

  e.g.

```
>Math.rint(2.3); // = 2.0
>Math.rint(2.9); // = 3.0
>Math.rint(2.5); // = 2.0
>Math.rint(3.5); // = 4.0
```

- ***int round(float x)***

  Return (int)Math.floor(x+0.5).

  e.g.

```
>Math.round(2.3); // = 2
>Math.round(2.9); // = 3
>Math.round(2.5); // = 3
>Math.round(3.5); // = 4
```

Computing Mean and Standard Deviation

Generate 10 random integer numbers and compute the mean and standard deviation of the set.

$$mean = \frac{\sum_{i=1}^{n} x_i}{n} \qquad deviation = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean)^2}{n-1}}$$

ComputeMeanDeviation

Run

min, max, abs, and random

- `max(a, b)` and `min(a, b)`

  Returns the maximum or minimum of two parameters.

- `abs(a)`

  Returns the absolute value of the parameter.

- `random()`

  Returns a random `double` value
  in the range (0.0, 1.0).

Overloading Methods

Overloading – creating a new Method accepting different types (and optionally numbers) of arguments, but using the same name as an existing Method.

Overloading the `max` Method

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

TestMethodOverloading

Run

Ambiguous Invocation

- Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
- This is referred to as *ambiguous invocation*.
- Ambiguous invocation is a compilation error.

Ambiguous Invocation

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```