x++ vs. ++x

- x=y++
- x=++y

x=0;

a=++x;

b=x++;

What are the values of a, b, and x?

x++ vs. ++x

```
public class Plus{
    public static void main(String []args){
        int x=0;
        int a=++x;
        System.out.println(x);
        System.out.println(a);

        int b=x++;
        System.out.println(x);
        System.out.println(b);
    }
}
```

Output

1

1

2

1

Java Coding 4

Method madness

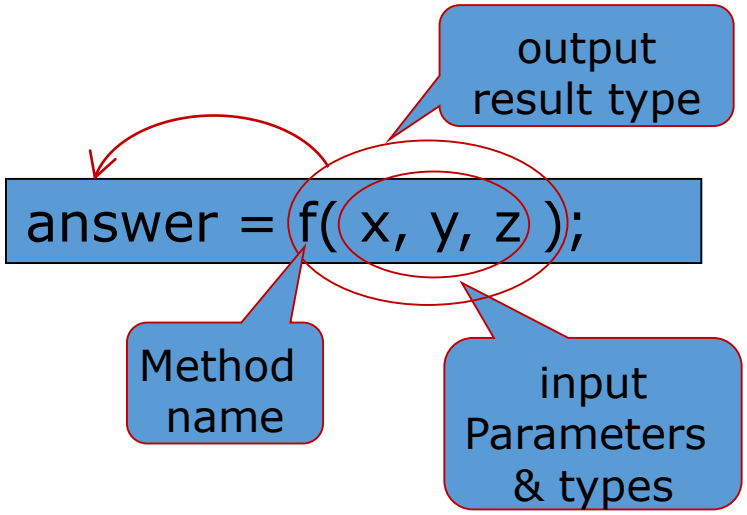
Using Java methods…

```
// in Scanner class
public  int nextInt()

int i = scan.nextInt();
```

```
// in … class
public  void println( String s)

System.out.println( “Hello Java”);
```

```
// in Math class
public static double sin( double d)

double d = Math.sin( 0.5);
```

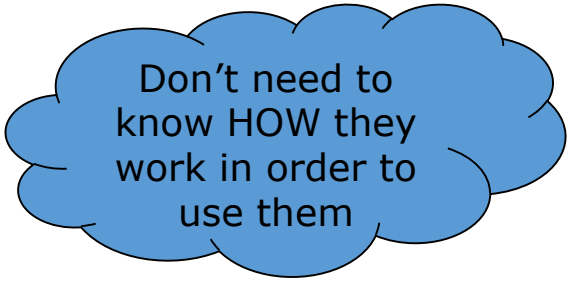
```
// in String class…
public String substring( int beginIndex, int endIndex)

String shortString = longString.substring( 1, 3);
```



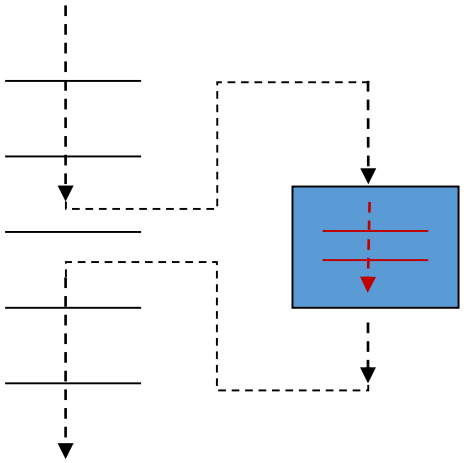
Use **ClassName.method(…)** for “static” methods

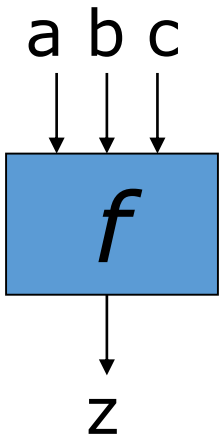
varName.method(…) for non-static methods

Methods



- Methods are
  - “Black” boxes that do something
- In Java
  - methods have any no of named inputs
  - “only” 1 output.





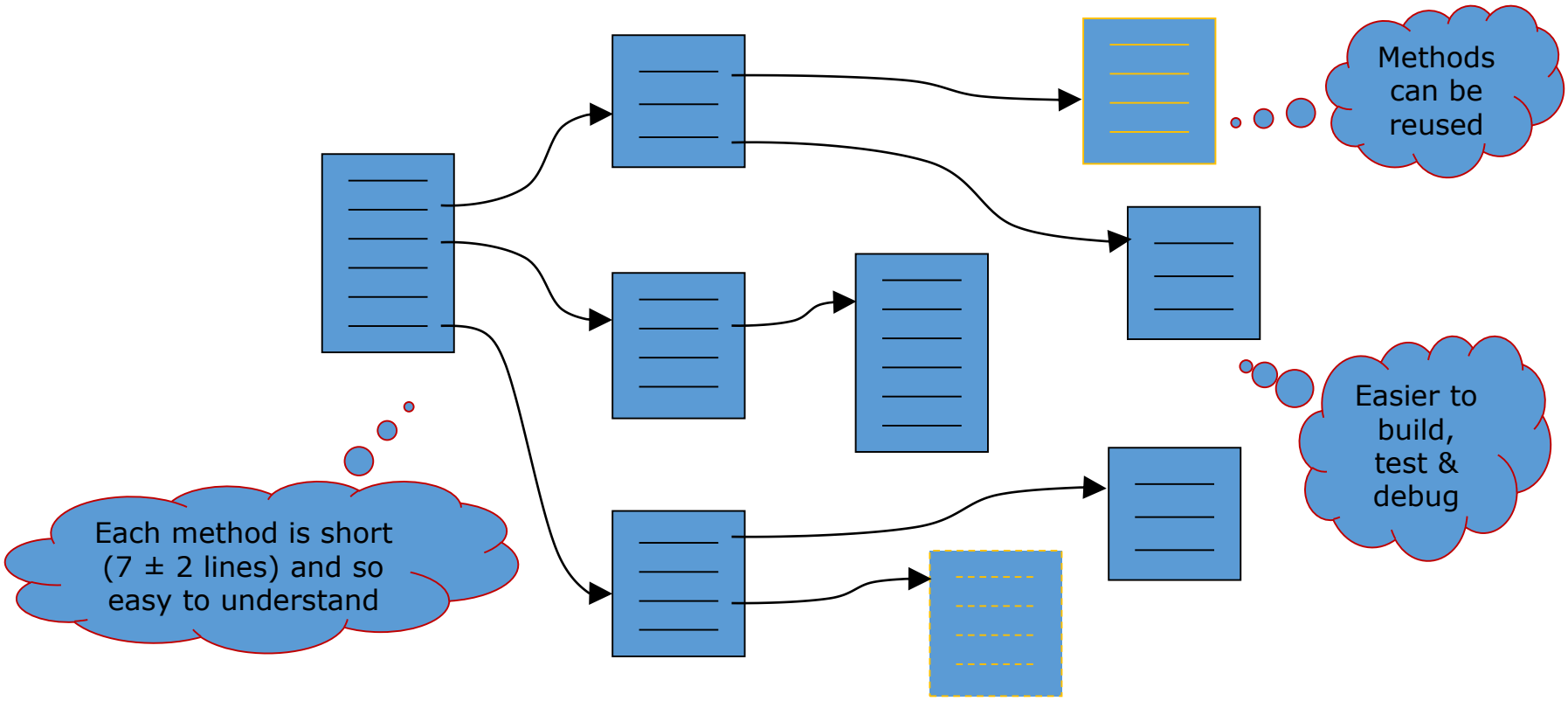
$$y = \sin(x)$$

functions ≡
methods in Java

$$z = f(a, b, c)$$

Methods

- Meaningfully named blocks of commands
  - facilitate: reuse, structure & top-down design.



Methods & Algorithms

- public static type **f**(type **a**, type **b**, type **c**)

Output type or void

Method name

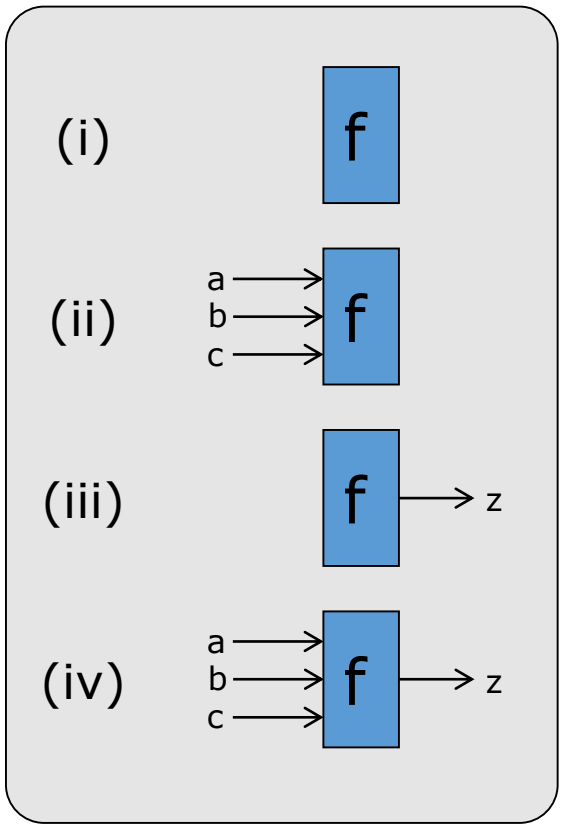
List of inputs & types

Ask for and get x (iii)

Compute y using x (iv)

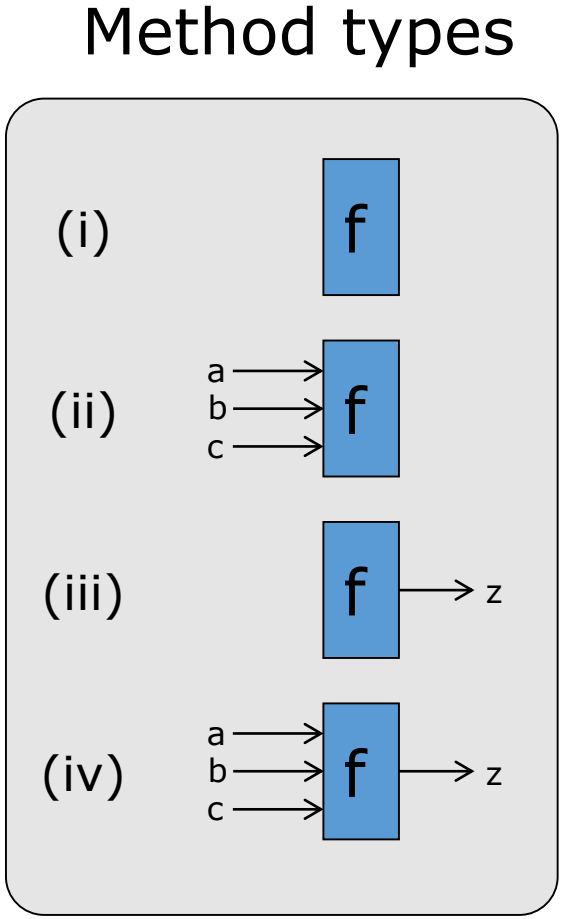
Print y (ii)

Method types



Methods & Algorithms

- Keyboard input & console (screen) output are not consider as input/output for methods.
  - “print y” has an input, but no output!
- What form is Java’s *main* method?
  - so far used as (i) but in fact it has a single input, args, so it is of type (ii)



Declaring Methods

- Syntax

```
modifiers resultType methodName( parameters)
    statement;
```

Formal parameters matched to actual parameter values at run-time

  - where
    - statement is any Java statement
    - methodName is identifier (convention as for variables)
    - resultType is any Java type or “void”
    - parameters is comma separated list of 0 or more “type name” pairs, eg. int age, String s, …

Modifiers

- Access
  - *coming next…*
- Others
  - final – cannot be changed!
  - static – no need for object, only one copy, can be accessed via class name.

```
y = Math.sin( x);
str = Integer.toString( 5, 2);
b = Character.isDigit( '3');
```

```
public static double sin( double value) {…}
public static String toString( int i, int radix) {…}
public static boolean isDigit( char ch) {…}
```

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

Modifiers

- In Java, methods grouped into classes, classes into packages. Packages can also contain other packages
- Math class contains methods sin, abs & sqrt
  - These methods must be defined public & static or else your program (class) couldn't use them!
- Math, Integer & Character are class names
  - All/some methods can be used without creating objects!
- These modifiers can also be used with data declarations
  - public static double PI = 3.142; is defined in the Math class, along with E

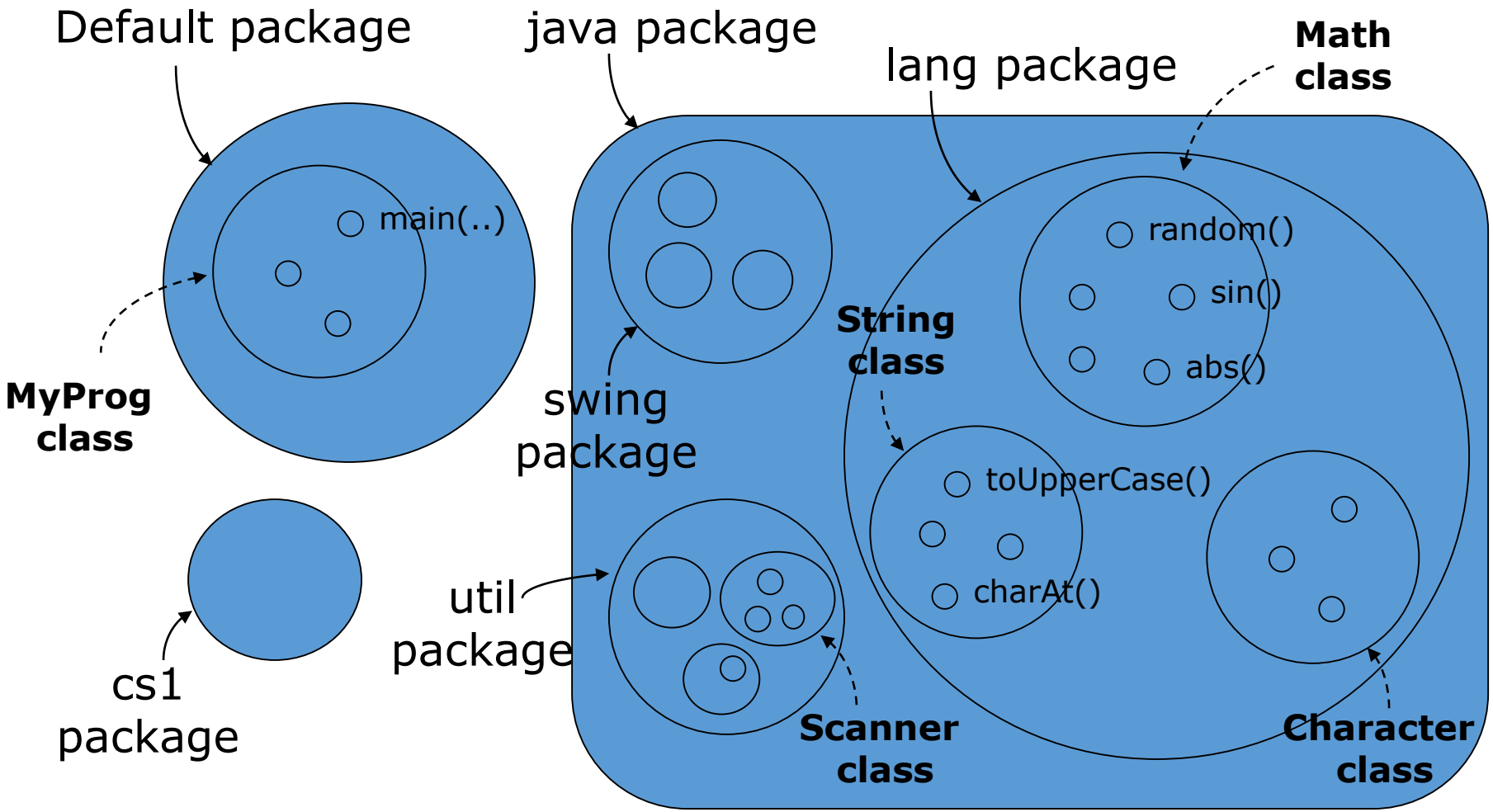
Modifiers

- Access
  - public – accessible/usable by all
  - protected – package & inherited
  - *default – package only* (when nothing is written)
  - private – only in current class

Methods are collections of statements
Classes are collections of methods
Packages are collections of classes
Packages can contain other packages

Need full "path" to class to use its methods
e.g. java.util.Scanner & java.lang.Math
mypackage.MyMethods

Packages, Classes & Methods



Packages, Classes & Methods

- Must specify full path to compiler
  - i.e. java.lang.Math, which is a pain so use import statement
- Scanner ***not static*** is in java.util package, hence we import java.util.Scanner in order to use it
- Math is in java.lang package which is imported by default so do not need explicit import
- Some classes contain only static methods, others only non-static, and other both!

Examples (1)

- Declare method to print fixed message

```
public static void showErrorMsg() {
   System.out.println( "Error, try again!");
}
```

- Use… *(from inside same class)*

```
…
showErrorMsg();
…
```

```
…
if ( x > 10)
   showErrorMsg();
else
   …
…
```

```
…
while ( !done) {
   showErrorMsg();
   …
}
…
```

- Use… *(from outside class)*

```
mypackage.MyClassName.showErrorMsg();
```

Examples (2)

- Method to print any message in box

```
public static void showMsg( String msg) {
   System.out.println( "************" );
   System.out.println( "  " + msg);
   System.out.println( "************" );
}
```

- Use…

```
…
showMsg( "Hello");
…
```

```
…
if ( x > 10)
   showMsg( "too big!");
else
   showMsg( "ok!");
…
```

Examples (3)

- Method to simulate a die throw

```
public static int randomDieThrow() {
   return (int) (Math.random() * 6) + 1;
}
```

- Use…

```
…
int faceValue;
faceValue = randomDieThrow();
if ( faceValue == 6)
   System.out.println( “free throw”);
…
```

Return statement

- Used to indicate result of method/function

```
return value;
```

- where value is
  - Literal
  - Variable or constant
  - Expression

- Type of value must match method return type!
- Execution of method stops immediately
- Can have multiple return statements

Return statement

- Used to indicate result of method/function

```
return value;
```

Example with multiple return statements:

```
---
if (…)
   return i;
---
return -1;
```

Examples (4)

- Method to find hypotenuse of triangle

```
public static double hypotenuse( double side1, double side2) {
   return Math.sqrt( side1 * side1 + side2 * side2);
}
```

- Use…

```
double z = hypotenuse( 3, 4);
```

Actual & formal parameters are matched one-to-one in sequence, e.g side1 = 3, side2 = 4

Types of actual and corresponding formal parameter must match!!

Definition & Use

- Static methods

Methods declared in class (but outside main)

Return statement indicates what value will be considered the result of the method (function)

Use in main or other method

```
// header
// Author, date

public class ClassName {

  public static void myMethod1( String s) {
      System.out.println( s);
  }

  private static int myMethod2( int i) {
     return i * 2 + 1;
  }

  public static void main( String[] args) {

     myMethod1( "Hello");

     int j = myMethod2( 5);
     System.out.println( j);
  }
}
```

Examples (5)

- Method to find absolute difference

```
public static int absDiff( int x, int y) {
    int z;
    z = x - y;
    if ( z < 0)
        z = -z;
    return z;
}
```

Any variables, like z, that are not parameters should be defined locally.

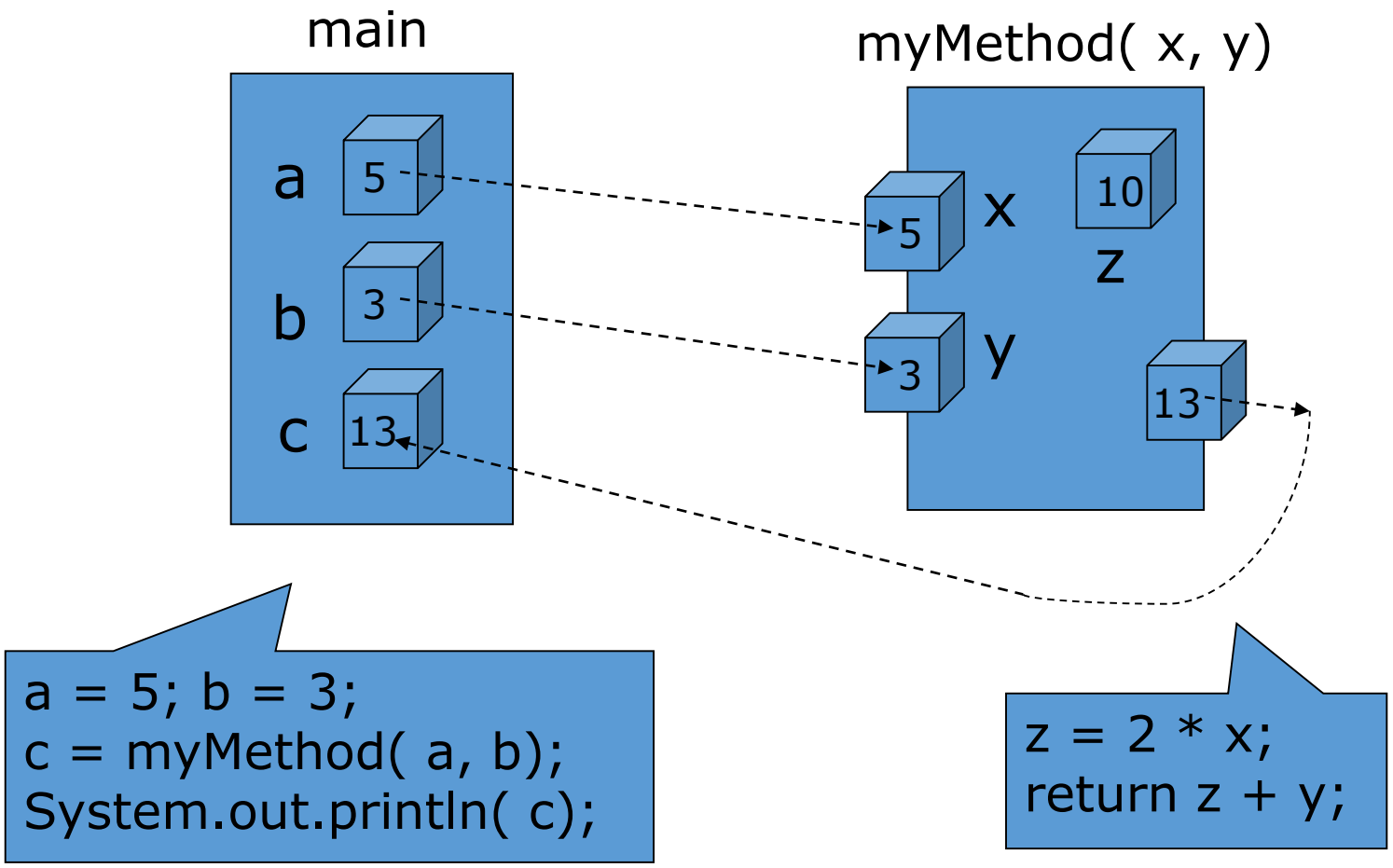
- Use…

```
int a = absDiff( 5, 3);
```

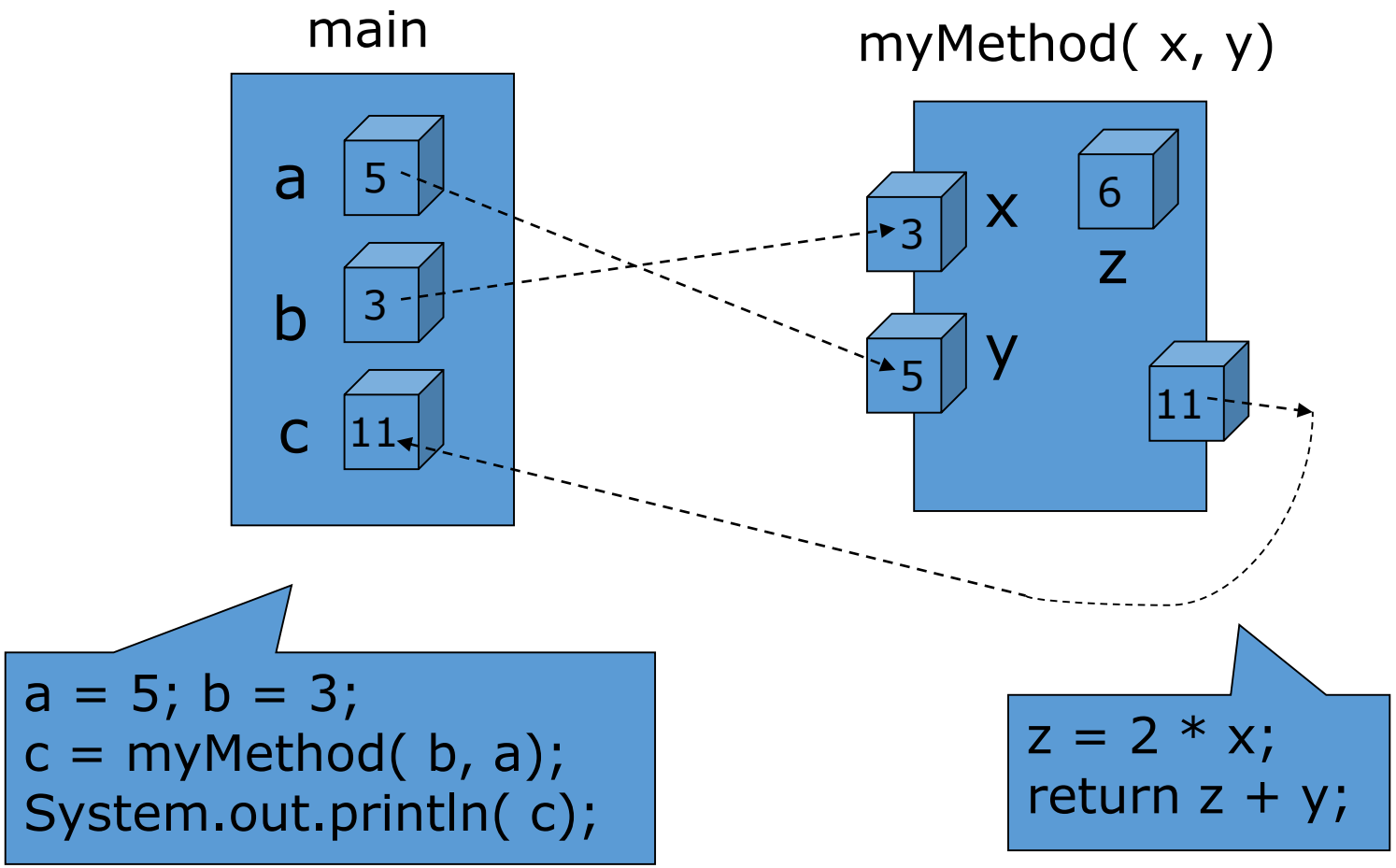
Actual & formal parameters are matched one-to-one in sequence, e.g x = 5, y = 3

Rewrite to avoid using the extra (temporary) variable z?

Parameter passing (1)



Parameter passing (2)



Parameter passing (2)

Notice that if inside myMethod we set x = 7; the corresponding actual parameter b does not change

Method exercises

- **isLessThan(x, y) - return boolean!**
- **Right justify string in given field width**
- **Return letter given grade**
- Determine if an int value isPrime or not (in class)
- isPalindrome (in class)
- **x to the power y**
- **Convert binary string to int**

isPrime

- Write a method to determine if an int value isPrime or not
- In the main method: isPrime(number)
- Also write the main method

isPrime

```
public static boolean isPrime(int num) {




}
```

isPrime (1)

```
public static boolean isPrime(int num) {

        for (int i = 2; i <= num-1){
            if (num % i == 0)
                return false;
        }
        return true;
}
```

isPrime (2) – more efficient

```
public static boolean isPrime(int num) {
        if (num == 2 ){
                return true;
         }

         for (int i = 2; i <= (int) Math.sqrt (num)+1; i++){
            if (num % i == 0)
                return false;
         }
        return true;
}
```

isPalindrome

- A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward
- A man, a plan, a canal, Panama
- Amor, Roma

- Write a method to determine if a given string is a palindrome
  - Assume one word string (no spaces)

isPalindrome

```
public class Palindrom {

public static void main(String[] args) {

        String check = "reliefpfpfeiller";

        System.out.println(isPalindrom(check));

    }
```

isPalindrome (1)

```
public static boolean isPalindrom(String test){
    boolean palindrom = false;
    if(test.length%2 == 0){
        for(int i = 0; i < test.length/2; i++){
            if(test.charAt(i) != test.charAt(test.length-i-1){
                return false;
            }
            else{
                palindrom = true;
            }
        }
    }else{
        for(int i = 0; i < (test.length-1)/2; i++){
            if(test.charAt(i) != test.charAt(test.length-i-1){
                return false;
            }else{
                palindrom = true;
            }
        }
    }
    return palindrom;
}
```

isPalindrome (2)

```
public static boolean isPalindrom(String test){
        boolean palindrom = false;
        int length = test.length();
        String reverse = "";

        for ( int i = length - 1; i >= 0; i-- ){
                reverse = reverse + test.charAt(i);
        }

        if (original.equals(reverse)){
                palindrom = true;
        }

        return palindrom;
}
```

gcd

- Write a method to return the greatest common divisor (gcd) of two given numbers
- In the main method: gcd(x,y)

- Also write the main method

gcd

```
public static int gcd(int a, int b) {




}
```

gcd

```
public static int gcd(int a, int b) {
        while ((a != 0) && (b != 0))
      {
         if (a > b)
         {
            a = a - b;
         }
         else
         {
            b = b - a;
         }
      }
        return a+b;
}
```

Methods calling methods

- Permutation
  - public static int comb(int n, int r)
- Combination
  - public static int perm(int n, int r)
- Factorial
  - private static int fact(int n)

Methods calling methods

```
public class Probability
{
	public static int comb( int n, int r)
	{
		return fact(n) / ( fact(r) * fact(n-r)
);
	}

	public static int perm( int n, int r)
	{
		return fact(n) / fact(n-r);
	}

	private static int fact( int n)
	{
		int factn = 1;
		for ( int i = 2; i <= n; i++)
			factn = factn * i;
		return factn;
	}
}
```

- Methods can call other sub/helper methods
- Provides top-down decomposition & abstraction

Methods calling methods

- *Caution:* method can call itself

- Useful problem solving technique known as recursion!

Sum of 1 to N

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N
- Write a method to solve this problem

```
public int sum (int num)
{




}
```

Can you do it without using the loop?

Sum of 1 to N

- Consider the problem of computing the sum of all the numbers between 1 and any positive integer N
- This problem can be recursively defined as:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + N - 1 + \sum_{i=1}^{N-2} i$$

$$= N + N - 1 + N - 2 + \sum_{i=1}^{N-3} i$$

$$\vdots$$

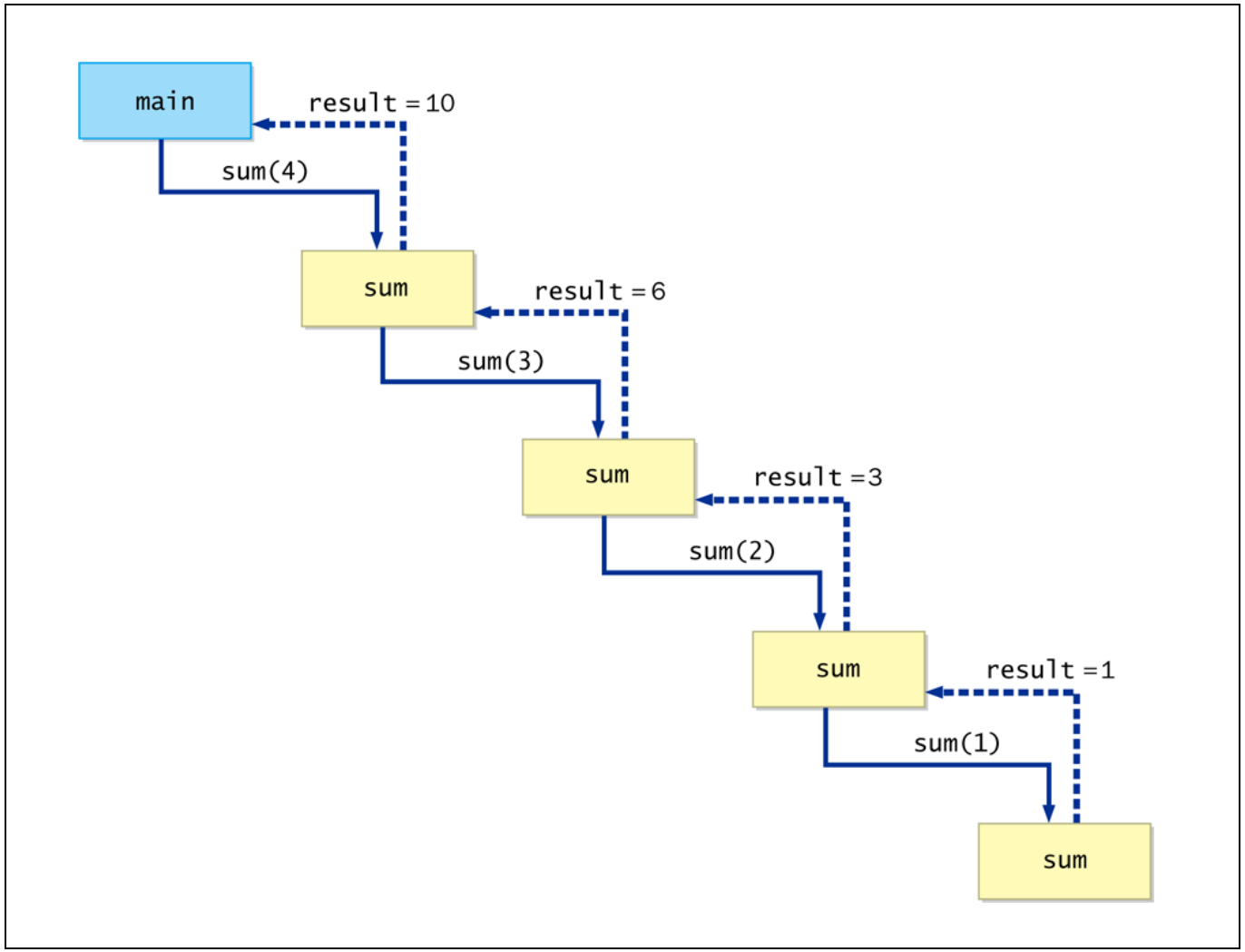
$$= N + N - 1 + N - 2 + \cdots + 2 + 1$$

Sum of 1 to N

- The summation could be implemented recursively as follows:

```
// This method returns the sum of 1 to num
public int sum (int num)
{
   int result;

   if (num == 1)
      result = 1;
   else
      result = num + sum (n-1);

   return result;
}
```

Sum of 1 to N


main
result = 10
sum(4)
sum
result = 6
sum(3)
sum
result = 3
sum(2)
sum
result = 1
sum(1)
sum

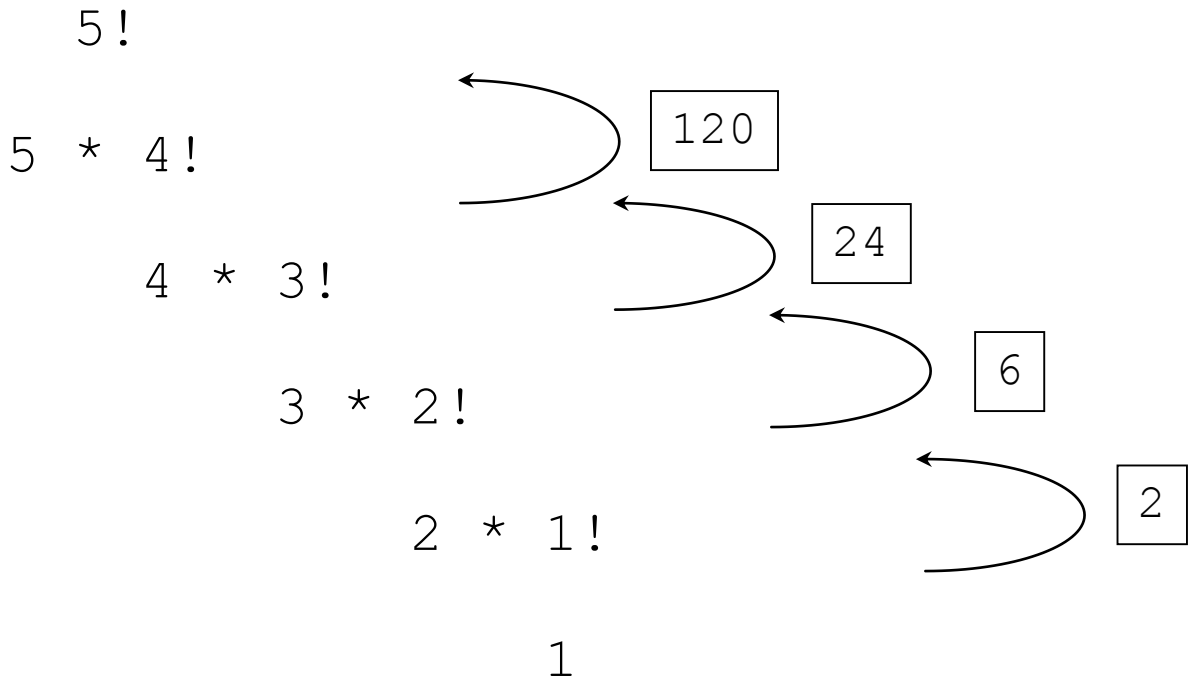
Recursive Factorial

- N!, for any positive integer N, is defined to be the product of all integers between 1 and N inclusive
- This definition can be expressed recursively as:

```
1!   =   1
N!   =   N * (N-1)!
```

- A factorial is defined in terms of another factorial
- Eventually, the base case of 1! is reached

Recursive Factorial



Recursive Factorial

```
public class FindFactorialRecursive
{
 public static void main (String args[])
 {
     System.out.println ("3! = " + factorial(3));
 }

 public static int factorial(int number)
 {
     if (( number == 1) || (number == 0))
          return 1;
     else
          return (number * factorial(number-1));
 }
}
```

Be aware

- Want methods to be reusable
- Methods that read inputs from keyboard or write output to console are difficult to reuse
- So, pass inputs as parameters
  & return the output as the result
- User interaction is then done elsewhere, usually main method.

Beware

```
a = 5;
b = 3;
d = 7;
c = myMethod( a, b);
Sys… ( a + “, ” + b + “, ” + d);
```

What is output?

Naturally assume method parameters are inputs only & method doesn’t have side effects!

Global variables!

What is the output of the following program?

```
public class finalA {
public static void main(String[] args) {
        int x = 3;
        int y = 9;
        method1(x, y);
        System.out.println("after method1:  " + x + "  " + y);

        x = 3;
        y = 9;
        x = method2(x, y);
        System.out.println("after method2:  " + x + "  " + y);
}

public static void method1(int x, int y){
        x = x + 2;
        y = x * 10;
}

public static int method2(int x, int y){
        y = y * 2;
        x = y - 5;
        return y;
}
}
```

after method1: 3 9
after method2: 18 9