Chapter 6: Methods

Yepang LIU

liuyp1@sustech.edu.cn

Objectives

- What is modular programming
- How to declare and use methods
- Method overloading

Problem Solving

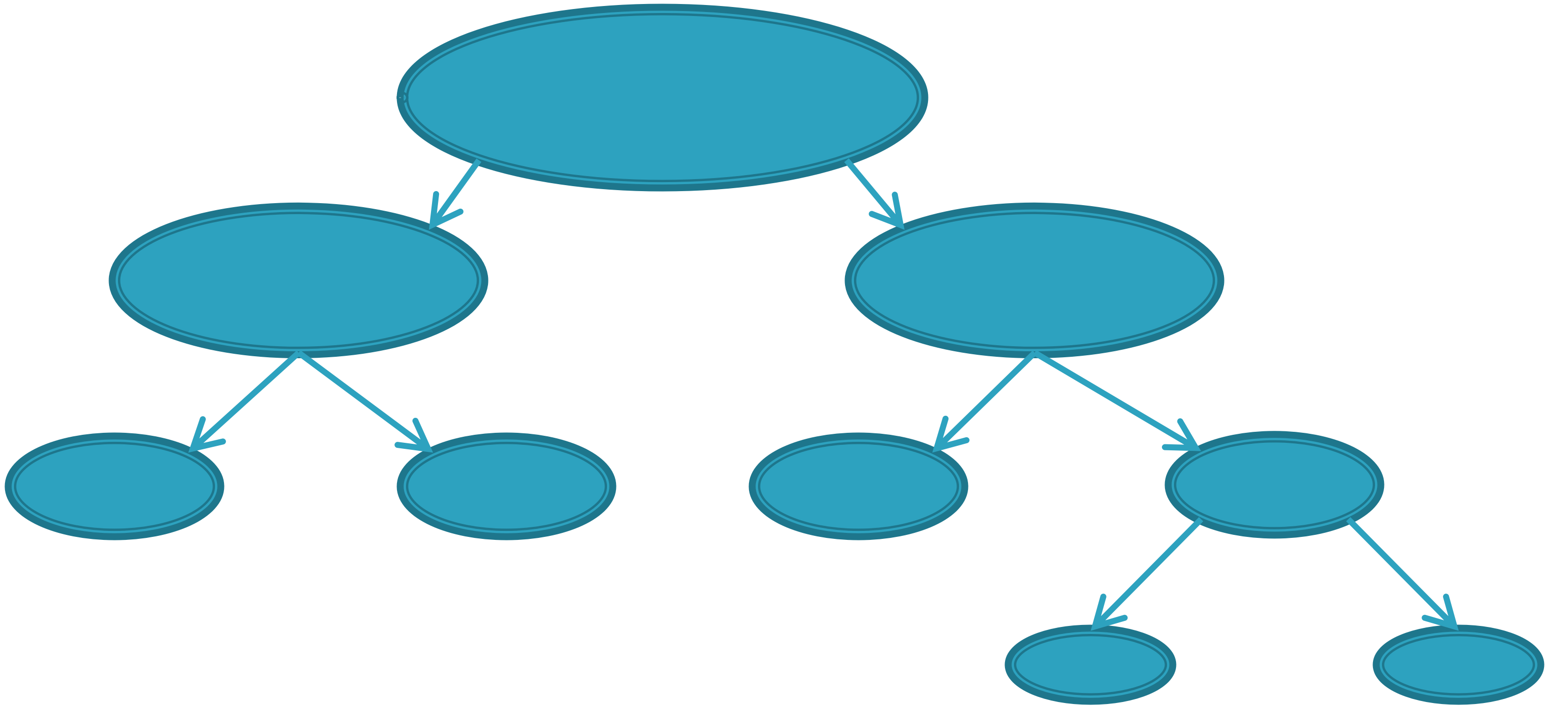
- The programs we have written so far solve simple problems (find the max in an array of numbers).
- They are short and everything fits well in a `main` method

What if you are asked to **solve complex problems**, e.g., building a climate model (气候模型) via analyzing big data?

Write a giant `main` method?

Divide and Conquer (分而治之)

- Decompose a big/complex task into smaller one and solve each of them



Methods

- Methods facilitate the design, implementation, operation and maintenance (维护) of large programs

Normally, methods are called on specific objects.

```
import java.util.Random;
public class NumberGuessing {
    public static void main(String[] args) {
        Random random = new Random();
        int magicNum = random.nextInt(10);
    }
}
```

Calling a method to generate a random number.
We don’t need to know how random numbers are generated.

Methods

- Static methods can be called without the need for an object of the class to exist

```
public static void main(String[] args) {
    double x1 = 3.14, y1 = -2.98;
    double x2 = -2.71, y2 = 7.15;

    System.out.println("The distance of the two points is " +
    Math.pow(Math.pow(x2-x1, 2) + Math.pow(y2-y1, 2), 0.5));
}
```

Static methods can be called directly using class names.

Why Use Methods?

- **For reusable code, reducing code duplication**
  - If you need to do the same thing many times, write a method to do it, then call the method each time you have to do that task.
- **To parameterize code**
  - You will often use parameters that change the way the method works.
- **For top-down programming (divide and conquer)**
  - You solve a big problem (the "top") by breaking it down into small problems. To do this in a program, you write a method for solving your big problem by calling other methods to solve the smaller parts of the problem, which similarly call other methods until you get down to simple methods that solve simple problems.

https://www.leepoint.net/JavaBasics/methods/method-commentary/methcom-purpose.html

Why Use Methods?

- **To create conceptual units**
  - Create methods to do something that is *one action* in your mental view of the problem. This will make it much easier for you to program.
- **To simplify**
  - Because local variables and statements of a method can not been seen from outside the method, they (and their complexity) are hidden from other parts of the program, which prevents accidental errors or confusion (e.g., random number generation method)
- **To ease debugging and maintenance**
  - You don’t want to debug a main method with 100K lines of code

Program Modules in Java

- Java programs are written by combining new methods and classes that you write with predefined methods and classes available in the framework and in various 3rd party libraries (第三方库)

- Related classes are typically grouped into packages so that they can be imported into programs and reused

- The Java API provides a rich collection of predefined classes (e.g., `java.util.Scanner`, `java.lang.Math`)

Program Modules in Java

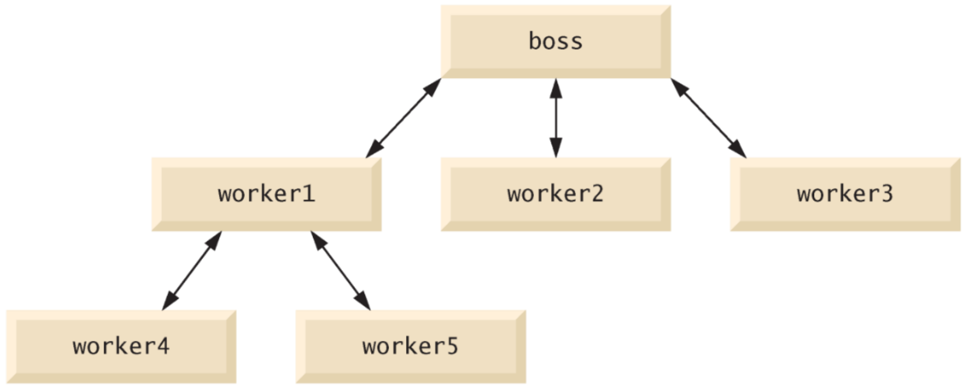
- A method is invoked by a method call
- When the called method (the callee) completes its task, it can return to the calling method (the caller):
  - Nothing (`void`, simply returning control back)
  - Primitive values (e.g., an integer)
  - References to objects, arrays

```
public static void main(String[] args) {
    Random random = new Random();
    int magicNum = random.nextInt(10);
}
```

Returns a primitive value to its caller: the `main` method

Program Modules in Java

- Similar to the hierarchical form of management
  - A boss (the caller) asks a worker (the callee) to perform a task and report back (return) the results after completing the task
  - The boss method does not know how the worker method performs its designated tasks (method complexity is hidden)
  - The worker may also call other worker methods, unknown to the boss



static Methods

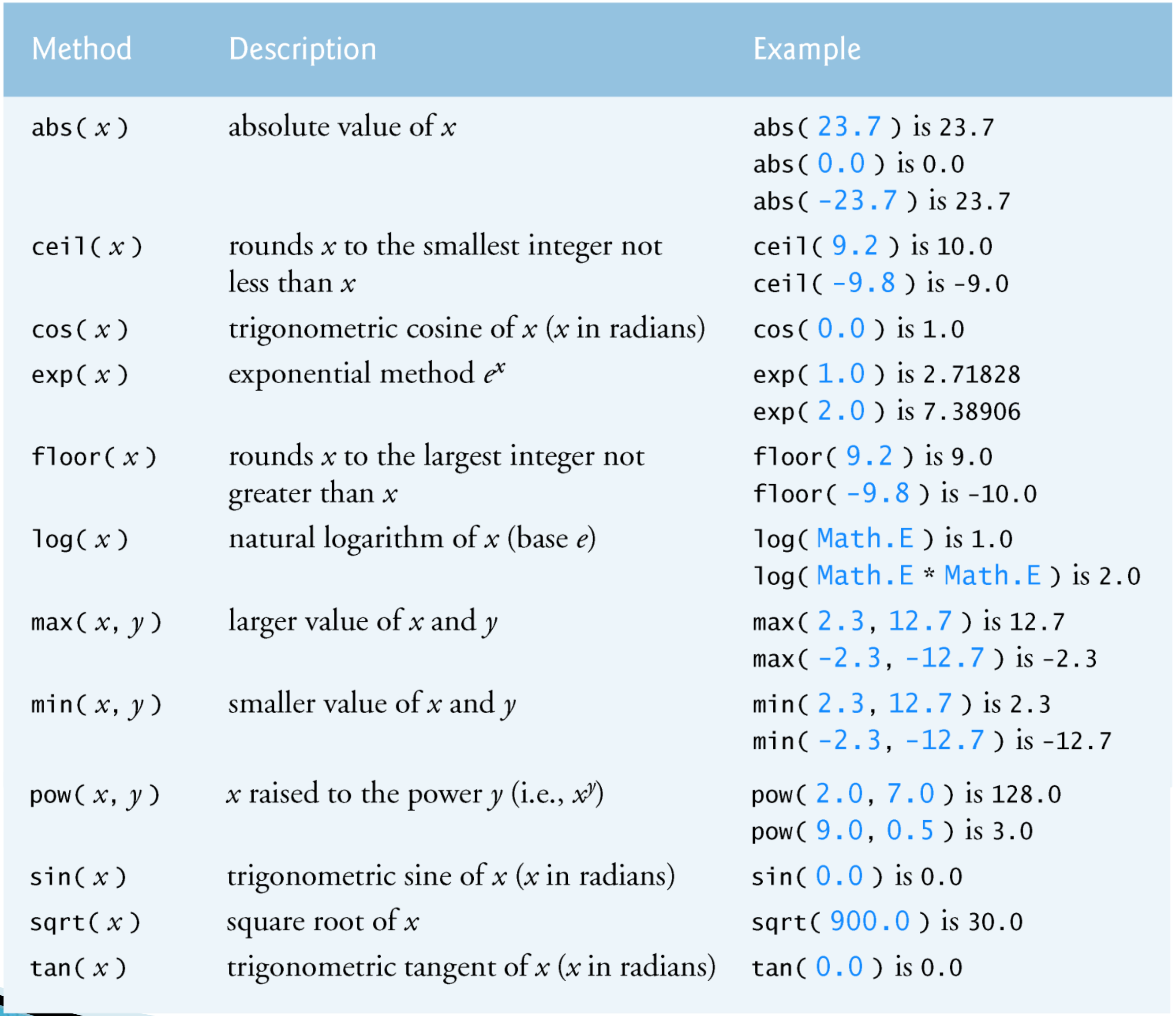
- Sometimes a method performs a task that does not depend on the contents of any object
  - Method applies to the class in which it’s declared
  - Known as a `static` method or a class method
  - Has the keyword `static` before the return type in the declaration
  - Called via the class name and a dot (`.`) separator

In `java.lang.Math` class:

```
public static double pow(double a, double b)
```

Returns the value of the first argument raised to the power of the second argument.

Many more useful static methods in `java.lang.Math` class:



Method	Description	Example

`abs( x )`	absolute value of x	`abs( 23.7 )` is `23.7`
		`abs( 0.0 )` is `0.0`
		`abs( -23.7 )` is `23.7`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`

`exp( x )`	exponential method e^x	`exp( 1.0 )` is `2.71828`
		`exp( 2.0 )` is `7.38906`

`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`log( x )`	natural logarithm of x (base e)	`log( Math.E )` is `1.0`
		`log( Math.E * Math.E )` is `2.0`

`max( x, y )`	larger value of x and y	`max( 2.3, 12.7 )` is `12.7`
		`max( -2.3, -12.7 )` is `-2.3`

`min( x, y )`	smaller value of x and y	`min( 2.3, 12.7 )` is `2.3`
		`min( -2.3, -12.7 )` is `-12.7`

`pow( x, y )`	x raised to the power y (i.e., x^y)	`pow( 2.0, 7.0 )` is `128.0`
		`pow( 9.0, 0.5 )` is `3.0`

`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Declaring Methods

The class defines two methods

```
import java.util.Scanner;
public class MaximumFinder {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("enter three floating-point values: ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();
        double result = maximum(number1, number2, number3);
        System.out.println("max is " + result);
    }

    public static double maximum(double x, double y, double z) {
        double max = x;
        if(y > max) max = y;
        if(z > max) max = z;
        return max;
    }
}
```

Find the largest of 3 double values

Details of Methods

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("enter three floating-point values: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();
    double result = maximum(number1, number2, number3);
    System.out.println("max is " + result);
}

public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

You need to call it explicitly to tell it to perform its task

Method don't get called automatically after declaration.

Details of Methods

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("enter three floating-point values: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();
    double result = maximum(number1, number2, number3);
    System.out.println("max is " + result);
}

public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Static methods in the same class can call each directly

Details of Methods

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("enter three floating-point values: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();
    double result = maximum(number1, number2, number3);
    System.out.println("max is " + result);
    double resultCeil = java.lang.Math.ceil(result);
}
```

Using static methods defined in other classes requires a fully qualified method name (全名，including the class name)

Details of Methods

Return type: the type of data the method returns to its caller. `void` means returning nothing.

```
public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Details of Methods

The method name follows the return type.
Naming convention: Lower Camel Case.

```
public static double maximum(double x, double y, double z) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Details of Methods

A comma-separated list of parameters, meaning that the method requires additional information from the caller to perform its task.

```
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Empty parentheses: the method does not need additional information to perform its task

Details of Methods

Each parameter must specify a type and an identifier

```
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

!!! A method's parameters are considered to be local variables of that method and can be used only in that method's body

Details of Methods

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("enter three floating-point values: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();
    double result = maximum(number1, number2, number3);
    System.out.println("max is " + result);
}

public static double maximum( double x, double y, double z ) {
    …
}
```

A method call supplies arguments for each of the method's parameters

One to one correspondence and the type must be consistent.

Details of Methods

Method header =

Modifiers + Return type + Method name + Parameters

```java
public static double maximum( double x, double y, double z ) {
    double max = x;
    if(y > max) max = y;
    if(z > max) max = z;
    return max;
}
```

Method body contains one or more statements that perform the method's task

The return statement returns a value (or just control) to the point in the program form which the method is called

Details of Methods

- Before any method can be called, its arguments must be evaluated to determine their values
- If an argument is a method call, the method call must be performed to determine its return value

```
Math.pow( Math.pow(x2-x1, 2) + Math.pow(y2-y1, 2) , 0.5 );
```

First argument

Returning Results

- If the method does not return a result, control returns when the program flow reaches the method-ending right brace
- Or when the statement `return;` executes
- If the method returns a result, the statement
  - `return expression;`

  evaluates the *expression*, then returns the result to the caller

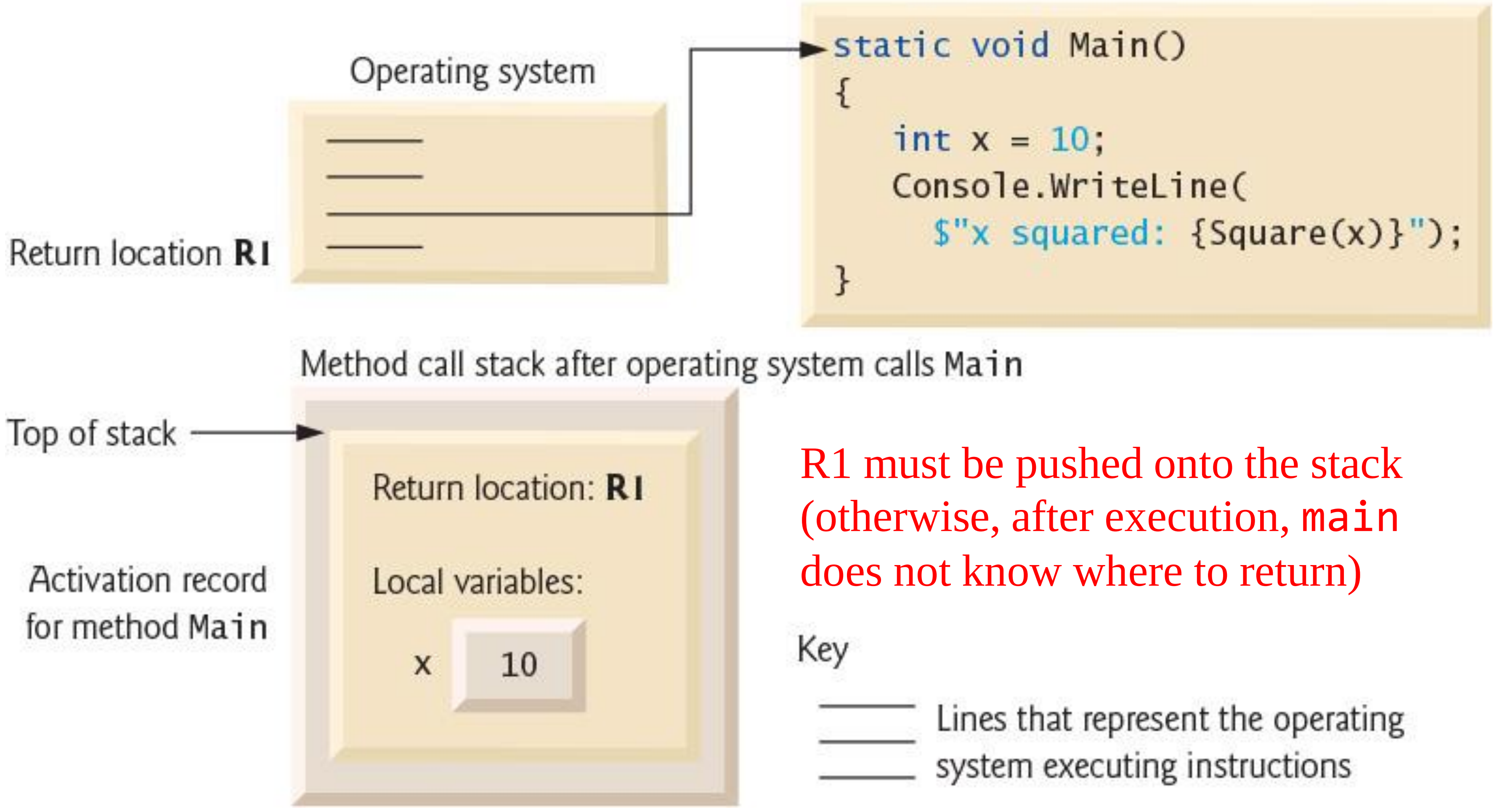
Method-Call Stack (方法调用栈)

- Stack data structure: analogous to a pile of dishes
  - When a dish is placed on the pile, it’s normally placed at the top (referred to as pushing onto the stack)
  - Similarly, when a dish is removed from the pile, it’s always removed from the top (referred to as popping off the stack)
- Last-in, first-out (LIFO) — the last item pushed (inserted) on the stack is the first item popped (removed) from the stack (also **first in last out**)

Method-Call Stack

- When a program calls a method, the called method must know how to return to its caller, so the **return address** of the calling method (next instruction to execute after method execution) is pushed onto the method-call stack (also known as program execution stack)

Step 1: Operating system calls `Main` to begin program execution


Operating system
Return location R1
Method call stack after operating system calls Main
Top of stack
Activation record for method Main
Return location: R1
Local variables:
x
10
Key
Lines that represent the operating system executing instructions

```
static void Main()
{
   int x = 10;
   Console.WriteLine(
     $"x squared: {Square(x)}");
}
```

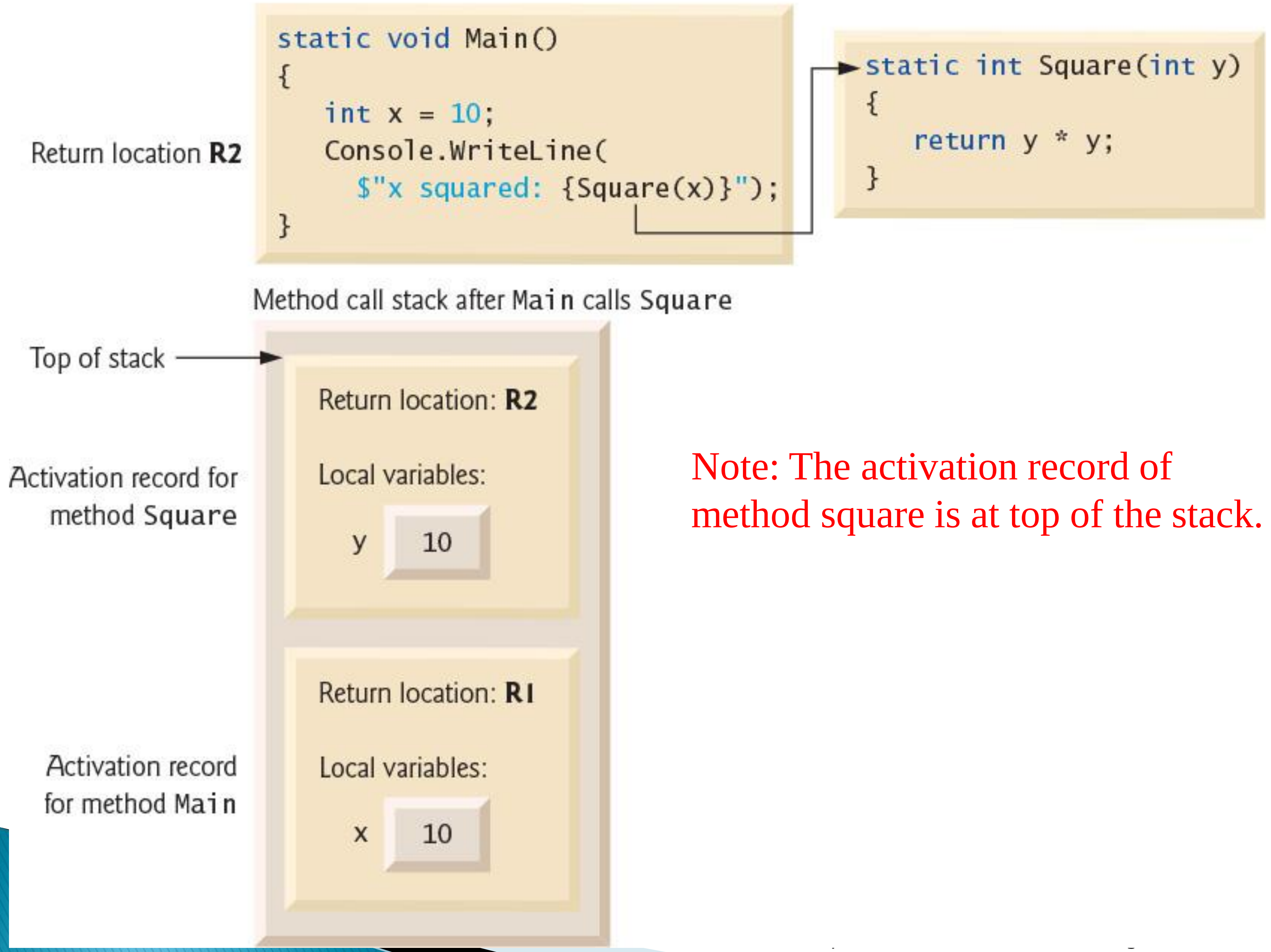
R1 must be pushed onto the stack (otherwise, after execution, `main` does not know where to return)

Method-Call Stack

- If a series of method calls occurs, the successive return addresses are pushed onto the stack in last-in, first-out order

- The program-execution stack also contains the memory for the local variables used in each invocation of a method
  - Stored in the activation record (or stack frame) of the method call
  - When a method call is made, the activation record for that method call is pushed onto the method-call stack

Step 2: Main calls method Square to perform calculation



```
static void Main()
{
    int x = 10;
    Console.WriteLine(
      $"x squared: {Square(x)}");
}
```

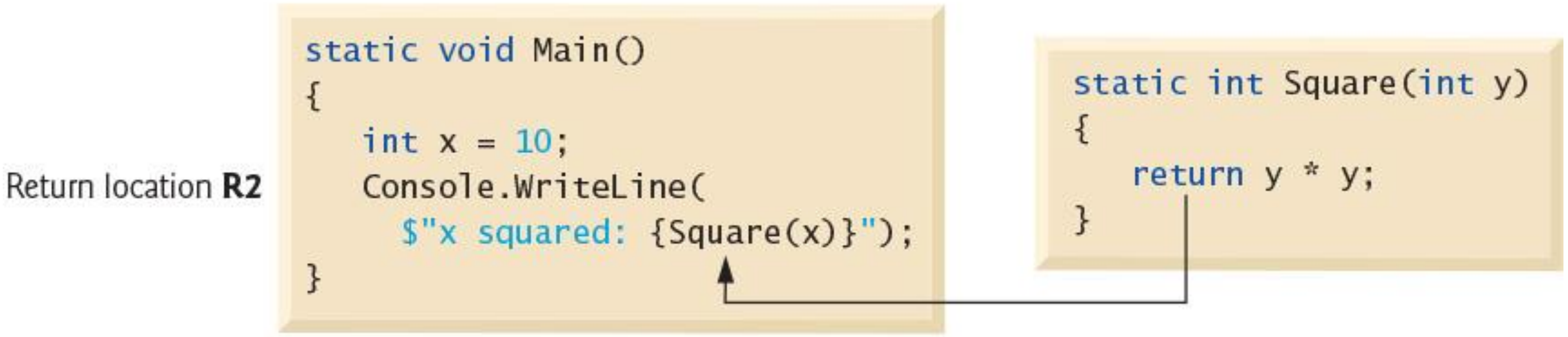
```
static int Square(int y)
{
    return y * y;
}
```

Note: The activation record of method square is at top of the stack.

Method-Call Stack

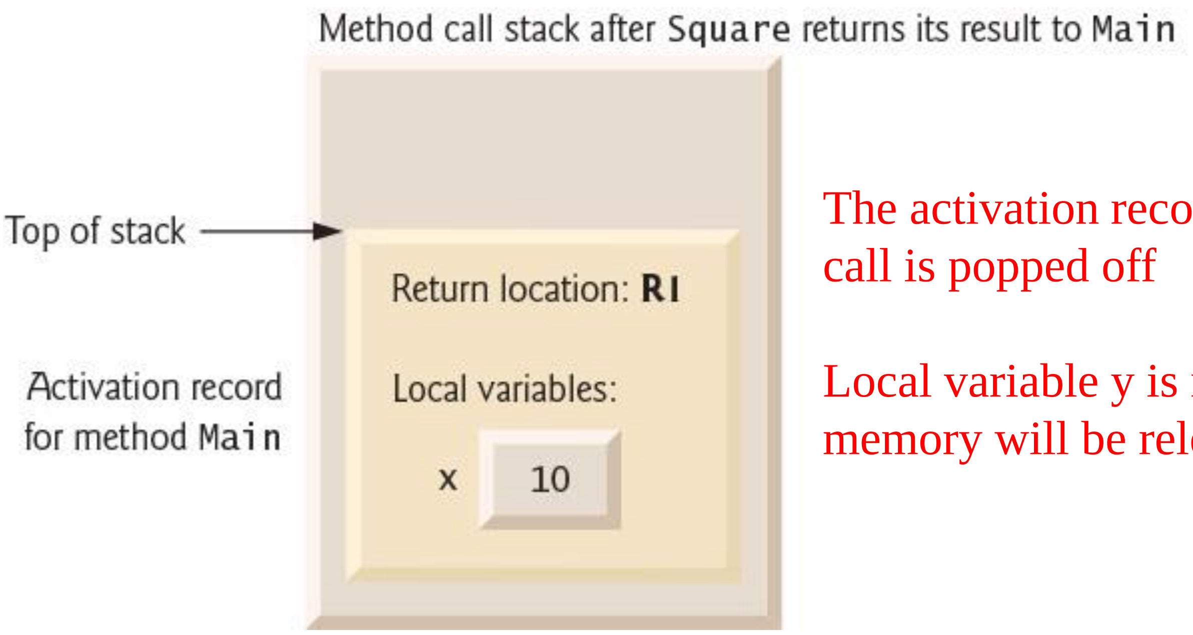
- When a method returns to its caller, the activation record for the method call is popped off the stack and those local variables are no longer known to the program

Step 3: Square returns its result to Main



```
static void Main()
{
   int x = 10;
   Console.WriteLine(
     $"x squared: {Square(x)}");
}
```

```
static int Square(int y)
{
   return y * y;
}
```


Method call stack after Square returns its result to Main
Top of stack
Activation record for method Main
Return location: R1
Local variables:
x
10

The activation record for the square method call is popped off

Local variable y is not visible anymore, its memory will be released for other uses

Passing Arguments in Method Calls

- Typically two ways: pass-by-value (值传递) and pass-by-reference (引用传递).
- When an argument is passed by value, a copy of the argument’s *value* is passed to the called method.
  - The called method works exclusively with the copy.
  - Changes to the copy do not affect the original variable’s value in the caller.
- When an argument is passed by reference (memory location), the called method can directly access the argument’s value in the caller and modify that data, if necessary.
  - Improves performance by avoiding copying possibly large amounts of data.

Pass-by-value in Java

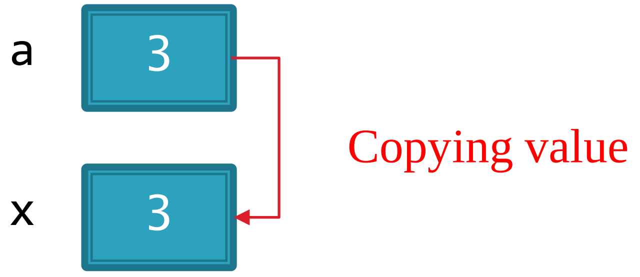
- In Java, all arguments are passed by value
- A method call can pass two types of values to the called method: copies of primitive values and copies of references to objects.
- Although an object's reference is passed by value, a method can still interact with the referenced object using the copy of the object's reference (arrays are also objects).
  - The parameter in the called method and the argument in the calling method refer to the same object in memory.

Example

```java
public static void main(String[] args) {
    int a = 3;
    System.out.println("Before: " + a);
    triple(a);
    System.out.println("After: " + a);
}

public static void triple(int x) {
    x *= 3;
}
```

```
Before: 3
After: 3
```



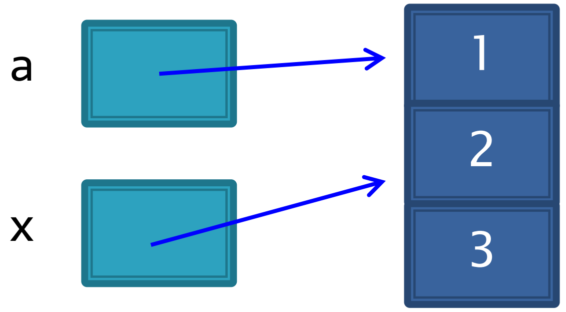
x becomes 9 after method call
a remains unchanged

Example

```
Before: 1 2 3
After: 3 6 9
```

```
public static void main(String[] args) {
    int[] a = {1, 2, 3};
    System.out.print("Before: ");
    for(int value : a) System.out.printf("%d ", value);
    triple(a);
    System.out.print("\nAfter: ");
    for(int value : a) System.out.printf("%d ", value);
}

public static void triple(int[] x) {
    for(int i = 0; i < x.length; i++)
        x[i] *= 3;
}
```



Copying value again. Difference is that the value is a memory address.

Argument Promotion

- Argument promotion—converting an argument’s value, if possible, to the type that the method expects to receive in its corresponding parameter

`Math.sqrt()` expects to receives a `double` argument, but it is ok to write `Math.sqrt(4)`: java converts the `int` value 4 to the `double` value 4.0

Promotion Rules

Specify which conversions are allowed (which conversions can be performed without losing data)

Type	Valid promotions
double	None
float	double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double (but not char)
byte	short, int, long, float or double (but not char)
boolean	None (boolean values are not considered to be numbers in Java)

Promotion Rules

Besides arguments passed to methods, the rules also apply to expressions containing values of two or more primitive types

```
2 * 2.0 becomes 4.0

int x = 2;
double y = x * 2.0;
// is x 2.0 or 2 now?
```

Answer: `x` is still of `int` type, the expression uses a temporary copy of `x`'s value for promotion

Method Overloading

- Methods of the same name can be declared in the same class, as long as they have different sets of parameters
- Used to create several methods that perform the same/similar tasks on different types or different numbers of arguments
  - Java compiler selects the appropriate method to call by examining the number, types and order of the arguments in the call

```
static double    max(double a, double b)
static float     max(float a, float b)
static int       max(int a, int b)
static long      max(long a, long b)
```

```
int a = 2;
int b = 3;
Math.max(a, b);
```

Which version of `max`?

Method Overloading

- Compiler distinguishes overloaded methods by their signature
  - A combination of the method’s name and the number, types and order of its parameters.

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
   //do the calculation here
   return 0.0;
}
```

```
Signature: calculateAnswer(double, int, double, double)
```

Method Overloading

- Method calls cannot be distinguished by return type. If you have overloaded methods only with different return types:
  - `int square(int a)`
  - `double square(int a)`
- and you called the method as follows
  - `square(2);`
- the compiler will be confused (since return value ignored)

Variable-Length Argument Lists

- With variable-length argument lists, you can create methods that receive an unspecified number of arguments.
- A type followed by an ellipsis (...) in a method's parameter list indicates that the method receives a variable number of arguments of that particular type.
  - `public static double average(double... numbers)`
  - Can occur only once in a parameter list, and the ellipsis, together with its type, must be placed at the end of the parameter list.
- Java treats the variable-length argument list as an array of the specified type.

Example

```java
public static double average(double... numbers) {
    double total = 0.0;
    for(double d : numbers) total += d;
    return total / numbers.length;
}

public static void main(String[] args) {
    double d1 = 10.0, d2 = 20.0, d3 = 30.0;
    System.out.printf("average of d1 and d2: %f\n", average(d1, d2));
    System.out.printf("average of d1 ~ d3: %f\n", average(d1, d2, d3));
}
```

```
average of d1 and d2: 15.000000
average of d1 ~ d3: 20.000000
```

Using Command-Line Arguments

- It’s possible to pass arguments from the command line to an application by including a parameter of type `String[]` in the parameter list of `main`.
  - `public static void main(String[] args)`
- By convention, this parameter is named `args`.
- When an application is executed using the `java` command, Java passes the command-line arguments that appear after the class name in the `java` command to the application’s `main` method as `Strings` in the array `args`.

Using Command-Line Arguments

```java
public class Example {
    public static void main(String[] args) {
        System.out.println("using command-line arguments");
        for(int i = 0; i < args.length; i++) {
            System.out.printf("argument %d: %s\n", i + 1, args[i]);
        }
    }
}
```

```
java Example
using command-line arguments
```

```
java Example hello world
using command-line arguments
argument 1: hello
argument 2: world
```

Java API Packages

- Java contains many predefined classes that are grouped into categories of related classes called packages
  - `java.lang`
  - `java.io`
  - `java.net`
  - `java.util`
- Overview of the packages in Java SE
  - https://docs.oracle.com/javase/8/docs/api/overview-summary.html