

Records and Methods

Introduction to Programming and
Computational Problem Solving:
Accelerated Pace

CSE 11

Lecture 2

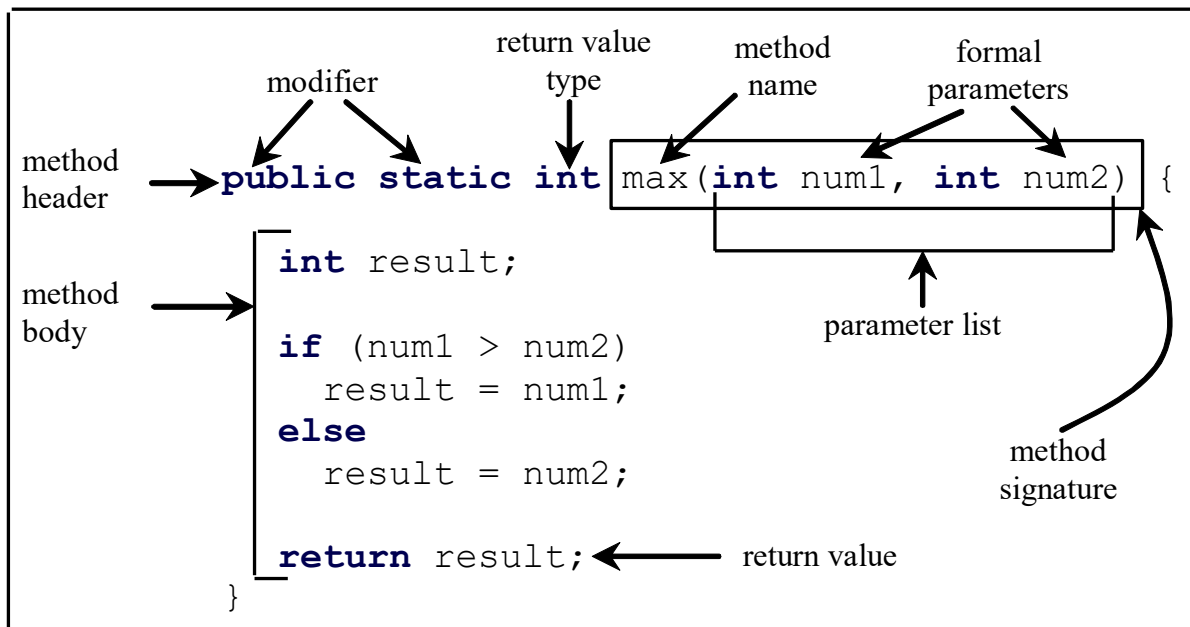
Variable and method names

- Naming convention: Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name
 - For example, the variables `radius` and `isLeapYear`, and the methods `kick` and `brake`.

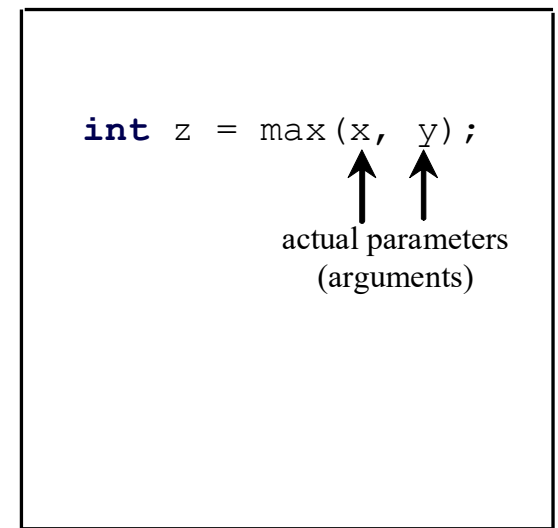
Defining methods

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

Define a method



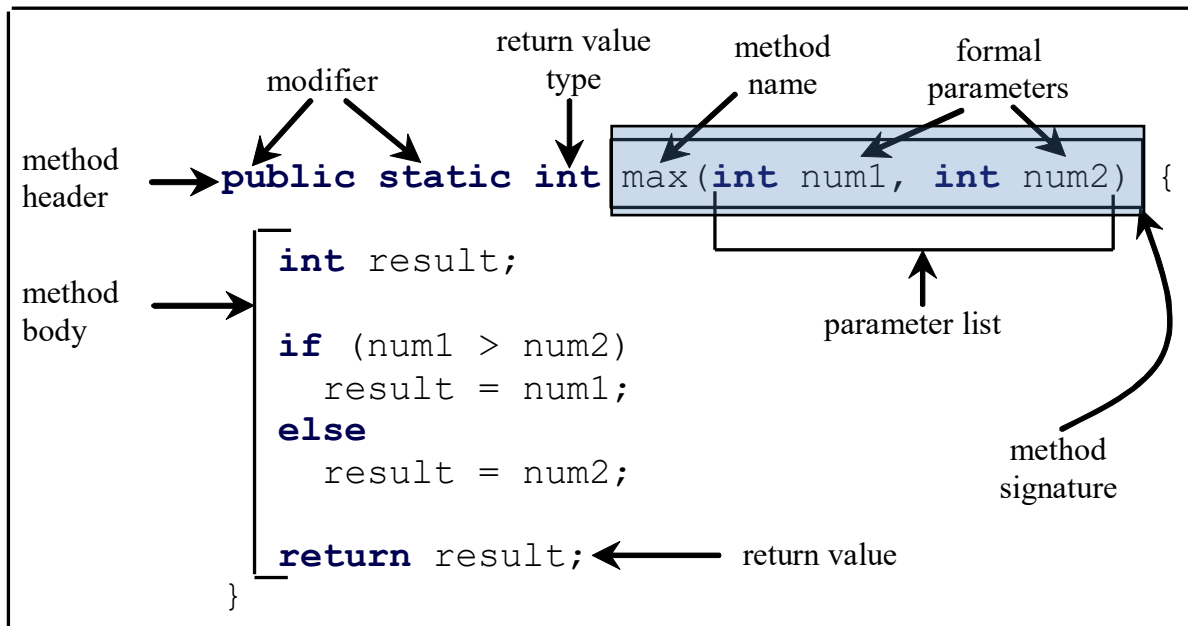
Invoke a method



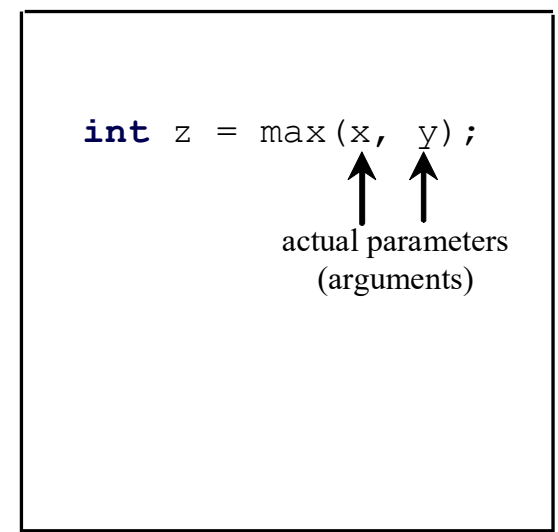
Method signature

- The *method signature* is the combination of the method name and the parameter list

Define a method



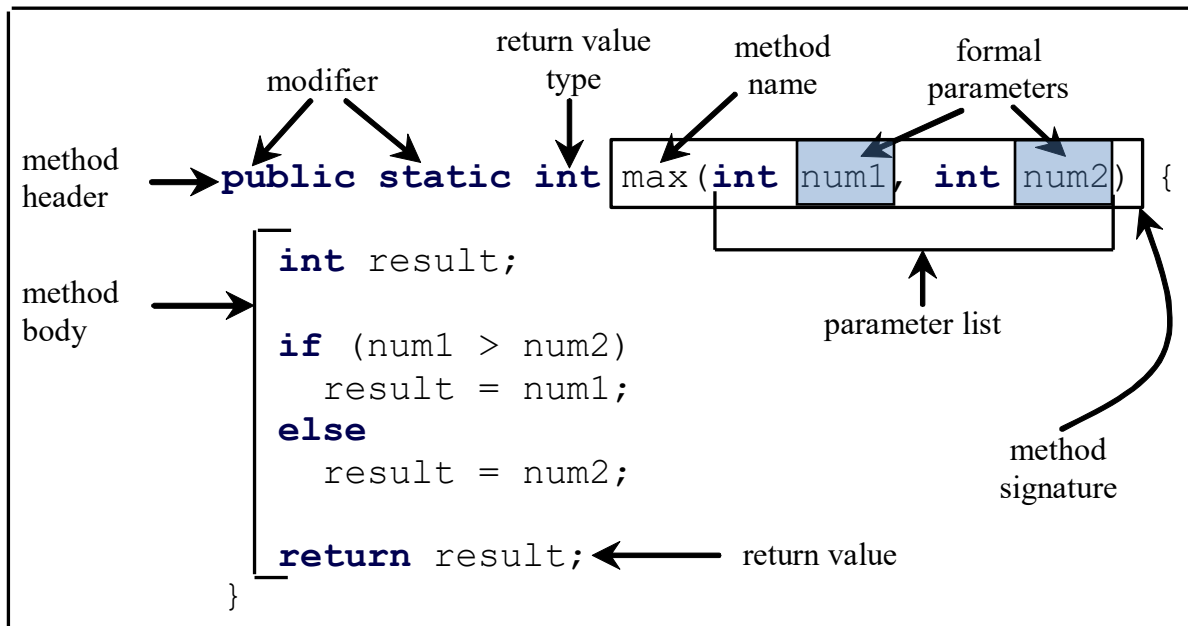
Invoke a method



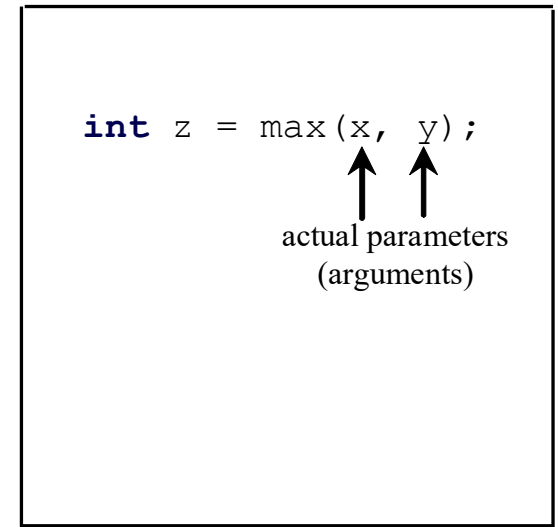
Formal parameters

- The variables defined in the method header are known as *formal parameters*

Define a method



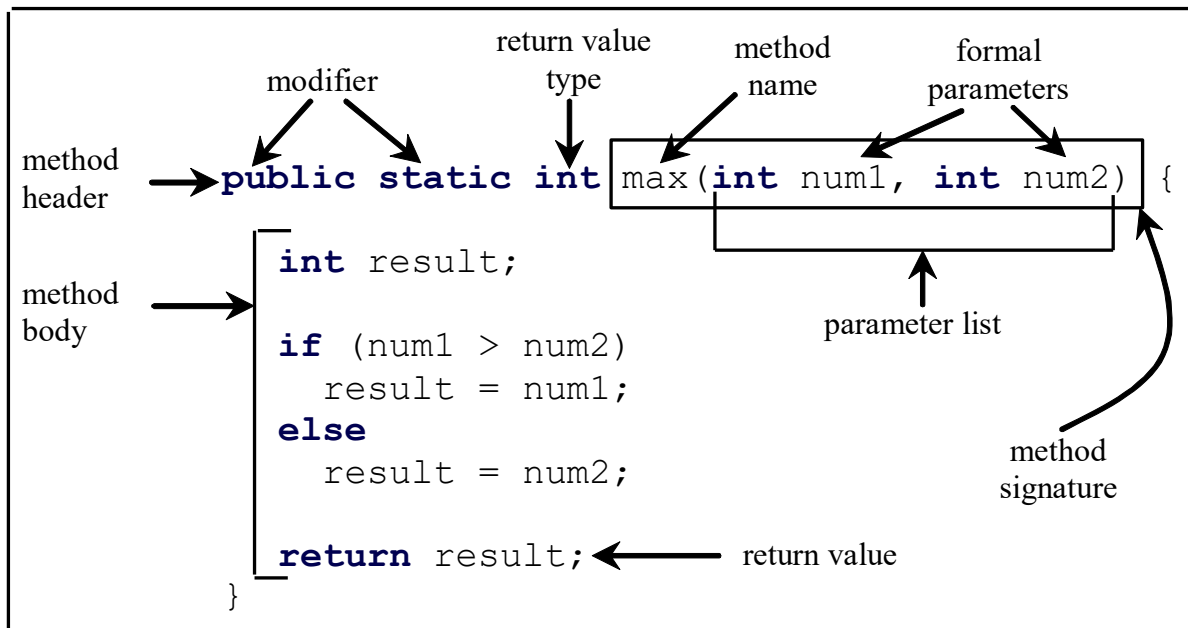
Invoke a method



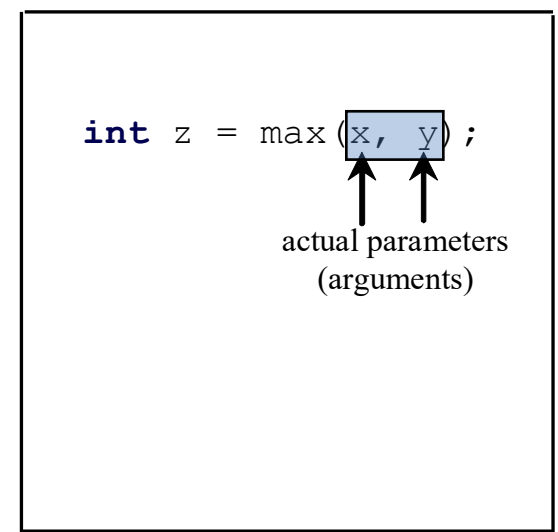
Actual parameters

- When a method is invoked, you pass a value to the parameter
 - This value is referred to as *actual parameter* or *argument*

Define a method



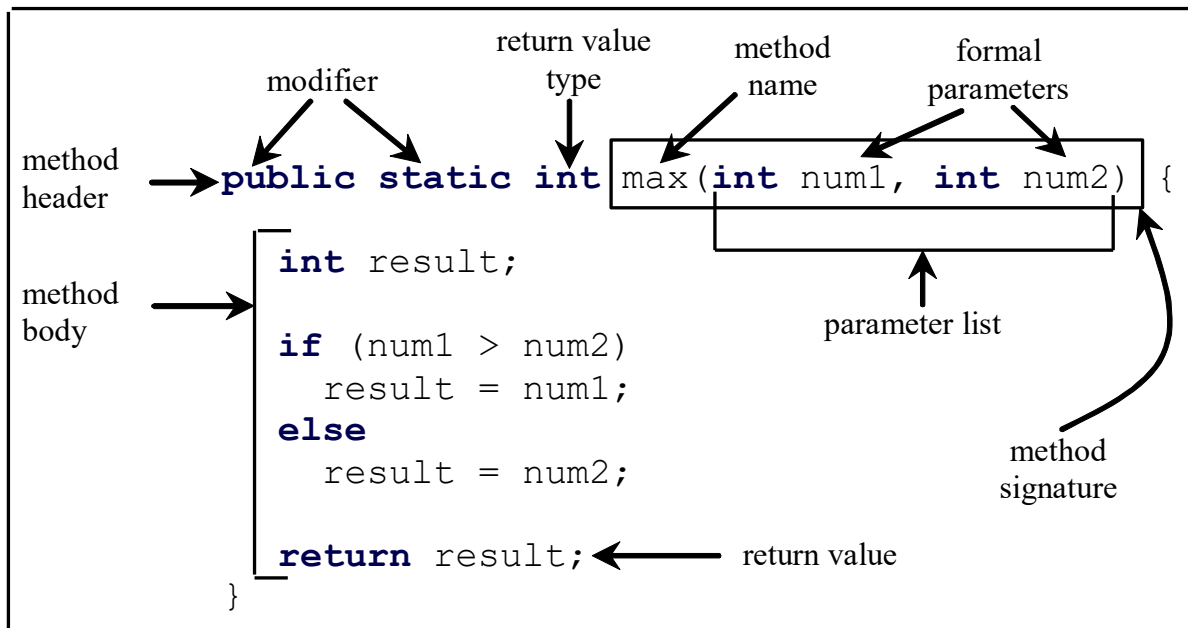
Invoke a method



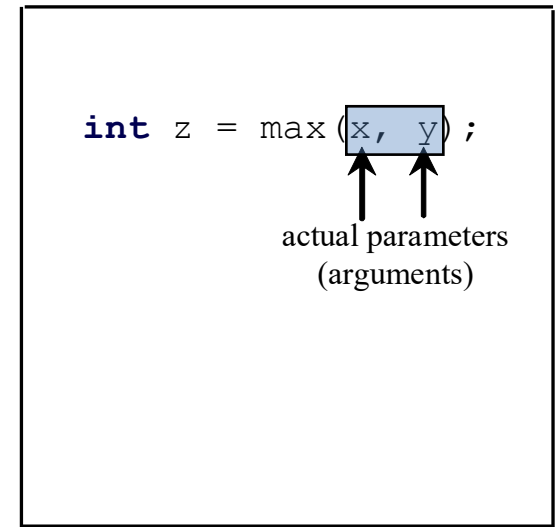
Pass by value

- Java uses **pass by value** to pass arguments to a method
- For example, modifying num1 does not modify x

Define a method



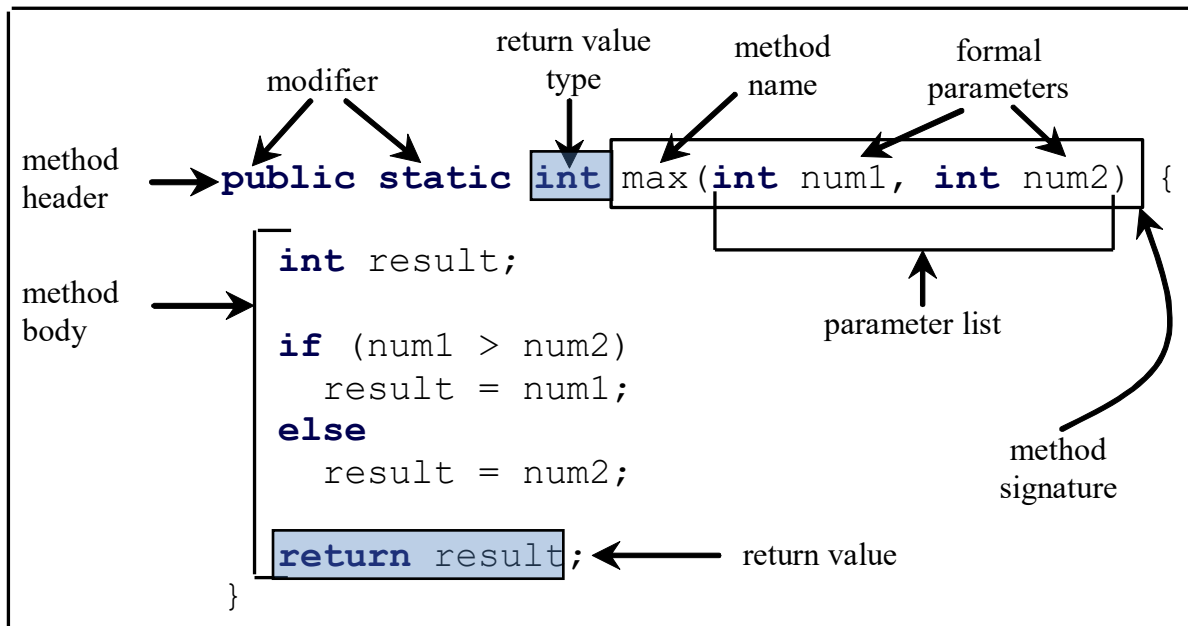
Invoke a method



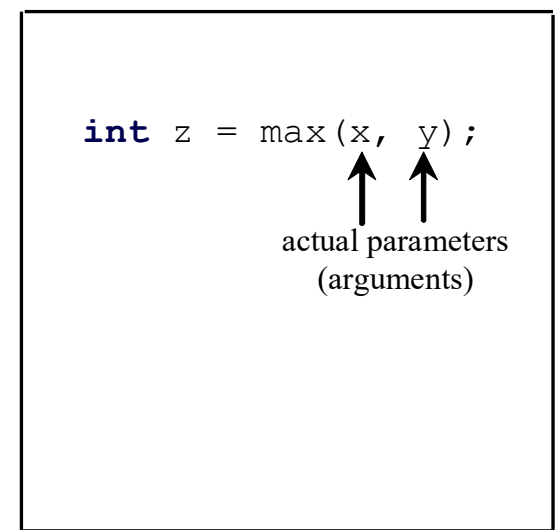
Return value type

- A method may return a value
- The *return value type* is the data type of the value the method returns
 - If the method does not return a value, the *return value type* is the keyword `void`

Define a method



Invoke a method



return statement

- A return statement is required for a value-returning method

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

(a)

Should be

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

(b)

Delete `if (n < 0)` in (a), so the compiler will see a return statement is reached regardless of how the `if` statement is evaluated

Trace code

i is declared and initialized

```
public static void main(String[] args) {  
    int i = 5;  
    int i = 2;  
    int k = max(i, i);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + i + " is " + k);  
}
```

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

i: 5

The main method
is invoked.

Trace code

j is declared and initialized

```
public static void main(String[] args) {
    int i = 5;
    int i = 2;
    int k = max(i, i);

    System.out.println(
        "The maximum between " + i +
        " and " + i + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

j: 2
i: 5

The main method
is invoked.

Trace code

Declare k

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

Space required for the
main method

k:
j: 2
i: 5

The main method
is invoked.

Trace code

Invoke max(i, j)

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i +
        " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Space required for the
main method

k:
j: 2
i: 5

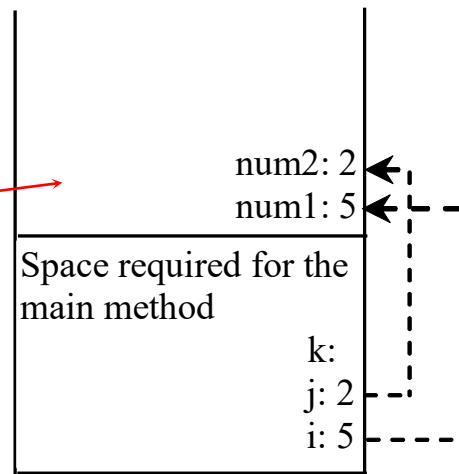
The main method
is invoked.

Trace code

pass the values of i and j to num1
and num2

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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



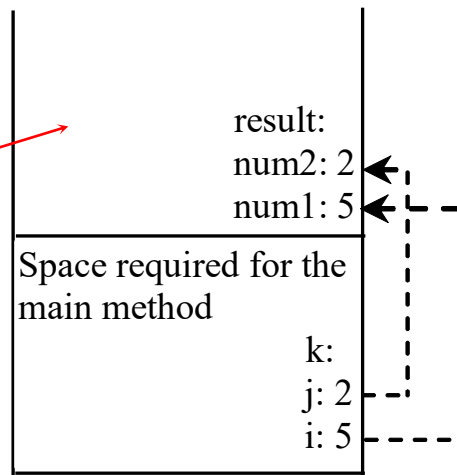
The max method is
invoked.

Trace code

Declare result

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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



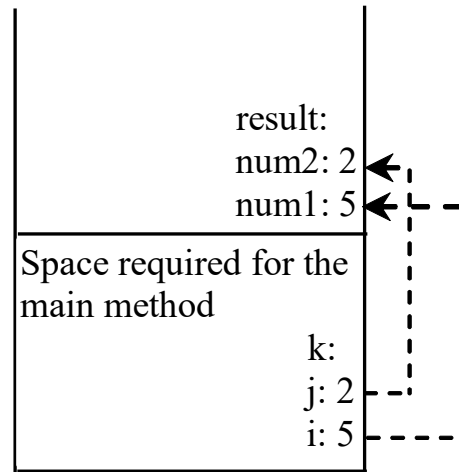
The max method is invoked.

Trace code

(num1 > num2) is true

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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



The max method is
invoked.

Trace code

Assign num1 to result

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

Space required for the
max method

result: 5
num2: 2
num1: 5

Space required for the
main method

k:
j: 2
i: 5

The max method is
invoked.

Trace code

Return result and assign it to k

```
public static void main(String[] args) {  
    int i = 5;  
    int j = 2;  
    int k = max(i, j);  
  
    System.out.println(  
        "The maximum between " + i +  
        " and " + j + " is " + k);  
}
```

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

Space required for the
max method

result: 5
num2: 2
num1: 5

Space required for the
main method

k: 5
j: 2
i: 5

The max method is
invoked.

Modularizing code

- Methods can be used to reduce redundant coding and enable code reuse
- Methods can also be used to modularize code and improve the quality of the program

Overloading methods

- Overloading methods enable you to define the methods with the same name **as long as their parameter lists are different**
- For example, overloading the max method

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

Ambiguous invocation

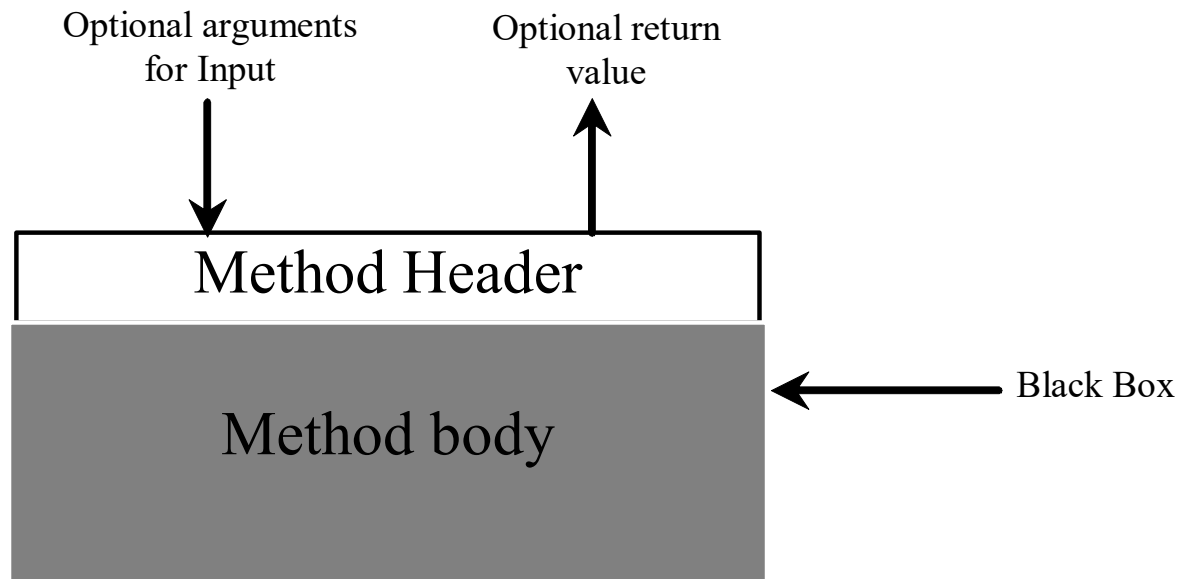
- The Java compiler determines which method to use based on the method signature
- Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match
- This is referred to as *ambiguous invocation*
- Ambiguous invocation is a compile error

Scope of local variables

- A local variable is a variable defined inside a method
- Scope is 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
- 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

Method abstraction

- You can think of the method body as a black box that contains the detailed implementation for the method (i.e., encapsulation)



Benefits of methods

- Write a method once and reuse it anywhere
- Information hiding
 - Hide the implementation from the user
- Reduce complexity

Objects and Java classes

- The state of an object consists of a set of data fields (also known as properties) with their current values
- The behavior of an object is defined by a set of methods
- A Java class uses variables to define data fields and methods to define behaviors

Records

- Java records act as concise, shallowly immutable data carriers that eliminate boilerplate code for simple classes
 - Automatic members: the compiler automatically generates private final fields, public accessor methods, a canonical constructor (i.e., a constructor that accepts all components of a record in the exact order and type they are declared), and standard implementations of `equals()`, `hashCode()`, and `toString()` (we'll cover these later in the quarter)
<https://docs.oracle.com/en/java/javase/25/language/records.html>
 - For example,

```
record Rect(int width, int height, String color) { }
```

Methods in a record class

- You can declare methods in a record class
- For example

```
record Rect(int width, int height, String color) {  
    boolean hasGreaterWidth(int compareWidth) {  
        return width > compareWidth;  
    }  
}
```

Accessing an object's members

- Use the **object member access operator**
 - Also called the **dot operator** (.)
- Reference the object's data using `objectRefVar.variableName`
- Invoke the object's method using `objectRefVar.methodName(arguments)`

Member variables and methods **do not use the dot operator** to access other member variables and methods **within the same class** (but, when method formal parameters have the same name as a member, then member variables and methods must be accessed a special way; covered later in the quarter).

Instance methods vs static methods

- Instance methods can only be invoked from a specific instance of a class
 - The syntax to invoke an instance method is
`referenceVariable.methodName(arguments)`
 - For example, nearly all of the `String` methods
<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/String.html>
`String message = "Welcome to Java";`
`int messageLength = message.length(); // Not String.length(message)`
- Static methods can be invoked without using an object (i.e., they are not tied to a specific instance)
 - All static methods are non-instance methods
 - The syntax to invoke a static method is
`ClassName.methodName(arguments)`
 - For example, all the methods defined in the `Math` class are static methods
<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/Math.html>
`double x = -1.2345;`
`double absx = Math.abs(x); // Not x.abs()`

Instance data fields and methods vs static data fields and methods

- **Instance** data fields and methods **can only be accessed using an object** (i.e., an instance of a class)
 - The syntax to access an **instance data field** is
`objectReferenceVariable.variableName`
 - The syntax to invoke an **instance method** is
`objectReferenceVariable.methodName(arguments)`
- **Static** data fields and methods (i.e., non-instance data fields and methods) can be accessed **without using an object** (i.e., they are not tied to a specific instance of a class)
 - The syntax to access a **static data field** is
`ClassName.variableName`
 - The syntax to invoke a **static method** is
`ClassName.methodName(arguments)`

Instance variables vs static variables

- An **instance** variable belongs to a specific instance of a class
- *A **static** variable is shared by all objects of the class*
 - Static variables are **shared** by all the instances of the class
 - Static constants are final variables **shared** by all the instances of the class

The `static` modifier

- To declare static variables, constants, and methods, use the `static` modifier
- `static` is a Java keyword

Named constants

- Naming convention: capitalize all letters in constants, and use underscores to connect words

```
static final datatype CONSTANT_NAME = VALUE;
```

```
static final int MAX_VALUE = 3;
```

```
static final char FIRST_UPPER_CASE = 'A';
```

Static members

- In code using a class, the best practice is to *make invocations of static methods and access of static data fields **obvious***
- Use
`ClassName.methodName(arguments)`
`ClassName.variableName`
- Do **not** use
`objectReferenceVariable.methodName(arguments)`
`objectReferenceVariable.variableName`

Limitations of static methods

- An **instance** method can
 - Invoke an **instance or static** method
 - Access an **instance or static** data field
- A **static** method can
 - Invoke a **static** method
 - Access a **static** data field
- A **static** method **cannot**
 - Invoke an **instance** method
 - Access an **instance** data field

Static methods

- If a member method or data field is independent of any specific instance, then make it static
- Do not require those using your class to create instance unless it is absolutely necessary

Next lecture

- `main` and Java