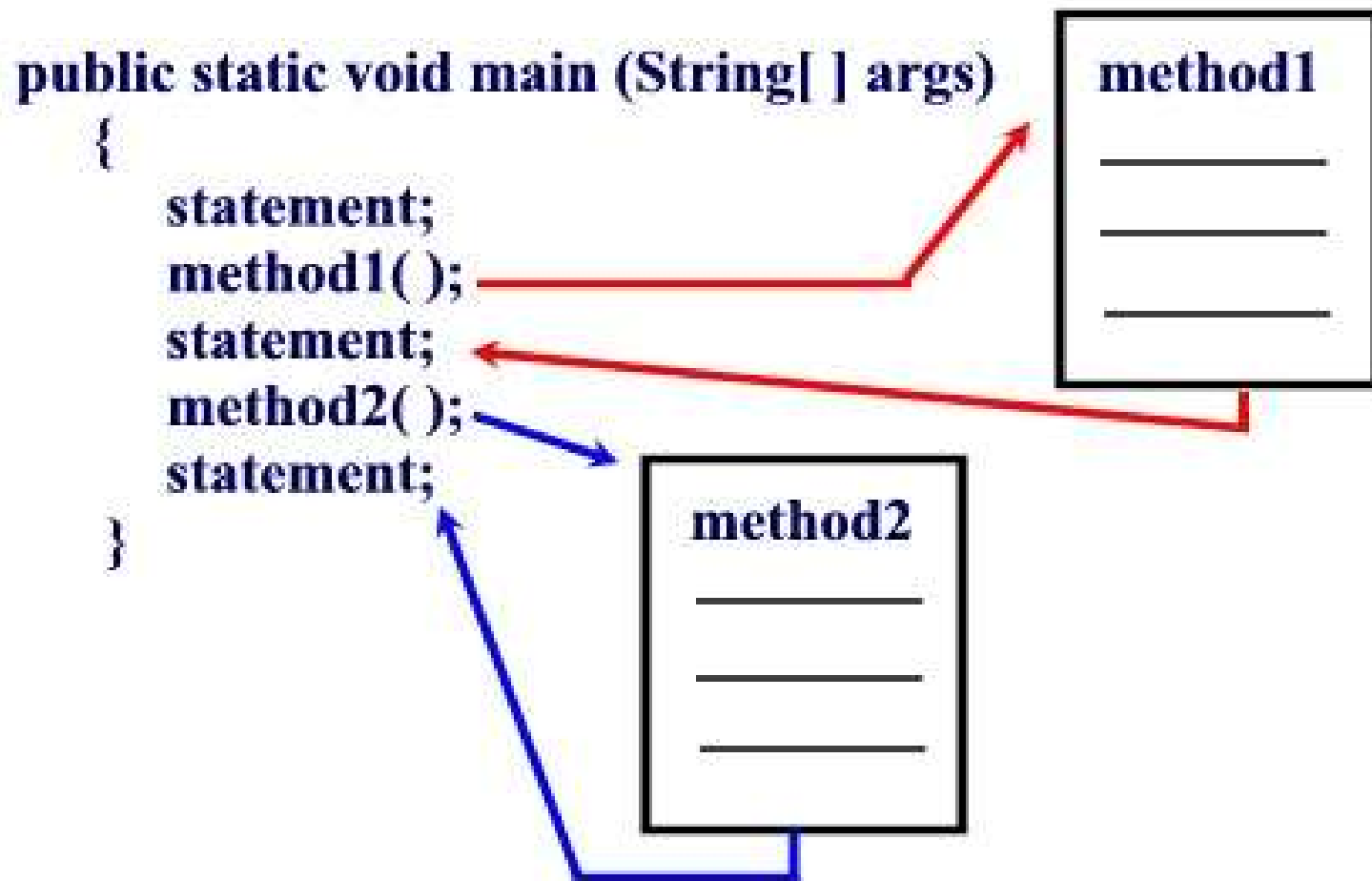# COMP 110/L Lecture 6

Mahdi Ebrahimi

Slides adapted from Dr. Kyle Dewey

Call Method
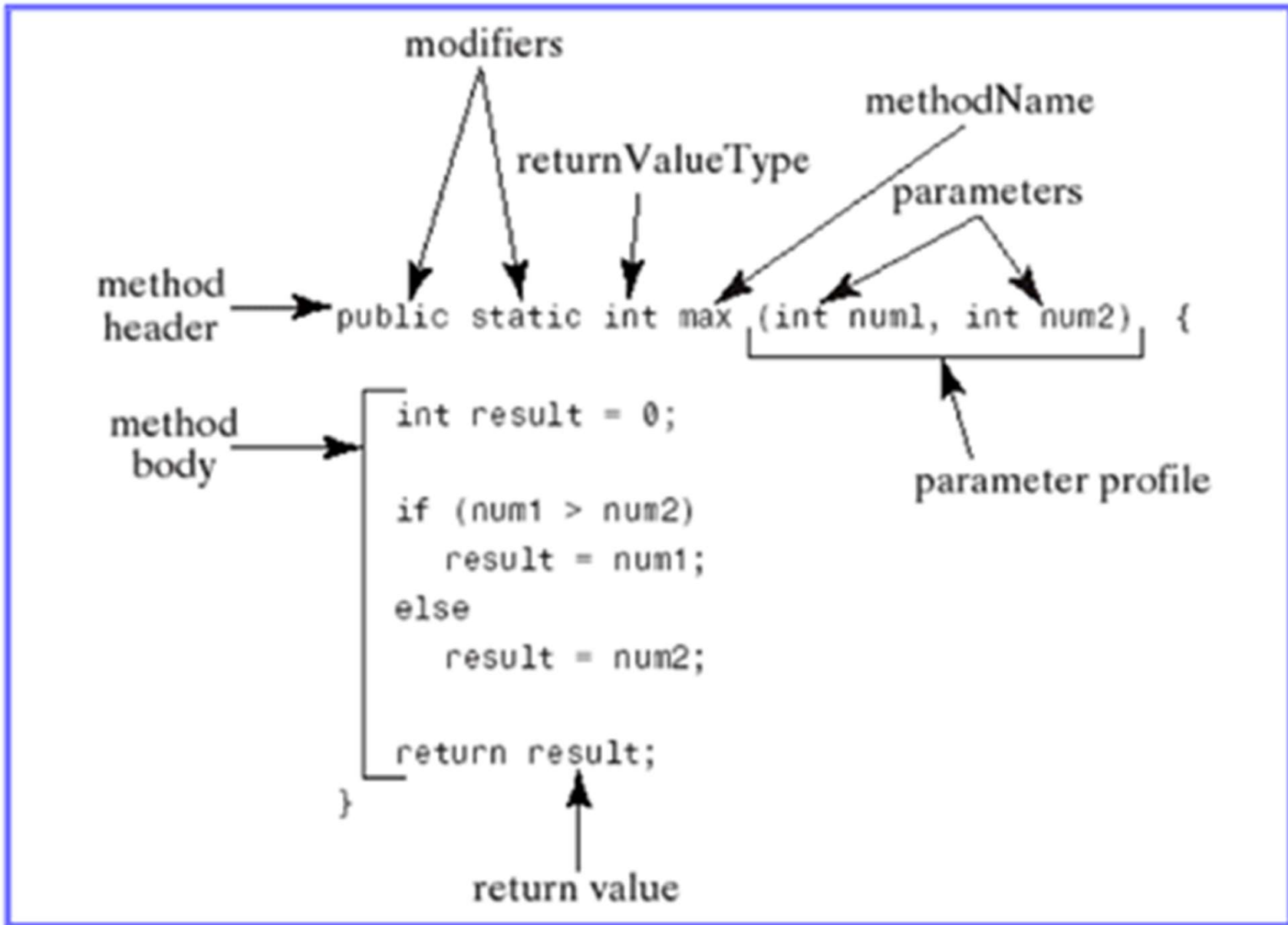
Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
  ...
  return expression;
}
```

Example

modifiers

returnValueType

methodName

parameters

method header → `public static int max (int num1, int num2) {`

parameter profile

method body →
```
int result = 0;

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

return result;
}
```

return value

Outline

- Methods
  - Variable scope
  - Call-by-value
- Testing with JUnit

Variable Scope

Question

Does this compile?

```java
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
        x = x + 1;
    }
}
```

Question

Does this compile?

```
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
```
Same name
```
}
```

Question

Does this compile?

```
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
}
```

Same name

Does not compile!

```
error: variable x is already defined in
method main
```

Methods and Variables

- Method parameters introduce new variables

- Method bodies may introduce new variables

Methods and Variables

- Method parameters introduce new variables

- Method bodies may introduce new variables

```
public  static int foo(int  x) {
  int  y = x + 1;

    return y;
 }
```

Methods and Variables

- Method parameters introduce new variables

- Method bodies may introduce new variables

```
public static int foo(int x) {
    int y = x + 1;
    return y;
}

public static void
main(String[] args) {
    int y = 8;
    System.out.println(y);
}
```

Methods and Variables

- Method parameters introduce new variables

- Method bodies may introduce new variables

```
public static int foo(int x) {
   int y = x + 1;
   return y;
}
```
Same name - does this compile?
```
public static void
main(String[] args) {
   int y = 8;
   System.out.println(y);
}
```

Methods and Variables

- Method parameters introduce new variables

- Method bodies may introduce new variables

```
public static int foo(int x) {
    int y = x + 1;
    return y;
}
public static void
main(String[] args) {
    int y = 8;
    System.out.println(y);
}
```

Same name - does this compile?

Yup!

Why?

- Declared variables have a *scope*

- Declaring two variables with the same name in the **same** scope:**error**

- Declaring two variables with the same name in **different** scopes:**OK**

- Scopes are introduced with { }

```java
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
}
```

```java
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
}
```

```
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
}
```

Scope of `main`

```
public class Test {
    public static void
    main(String[] args) {
        int x = 7;
        int x = 8;
    }
}
```

Same variable name in same scope: error

Scope of `main`

```java
public   static int  foo(int  x) {
   int   y = x + 1;
   return y;
}

public static void
main(String[] args) {
   int y = 8;
   System.out.println(y);
}
```

```java
public static int foo(int x) {
    int y = x + 1;
    return y;
}


public static void
main(String[] args) {
    int y = 8;
    System.out.println(y);
}
```

```
public static int foo(int x) {
    int y = x + 1;
    return y;
}
```

Scope of `foo`

```
public static void
main(String[] args) {
    int y = 8;
    System.out.println(y);
}
```

Scope of `main`

```
public static int foo(int x) {
    int  y  = x + 1;
    return y;
}
```

Scope of `foo`

Same variable name in different scopes:ok

```
public static void
main(String[] args) {
    int  y  = 8;
    System.out.println(foo(y));
}
```

Scope of `main`

Call-by-Value

Question

What does this code print?

```
public static int something(int x) {
   x = 1;
   return x;
}

public static void
main(String[] args) {
   int x = 8;
   something(x);
   System.out.println(x);
}
```

Question

What does this code print?
Answer:8

```java
public static void something(int x) {
    x = 1;
}

public static void
main(String[] args) {
    int x = 8;
    something(x);
    System.out.println(x);
}
```
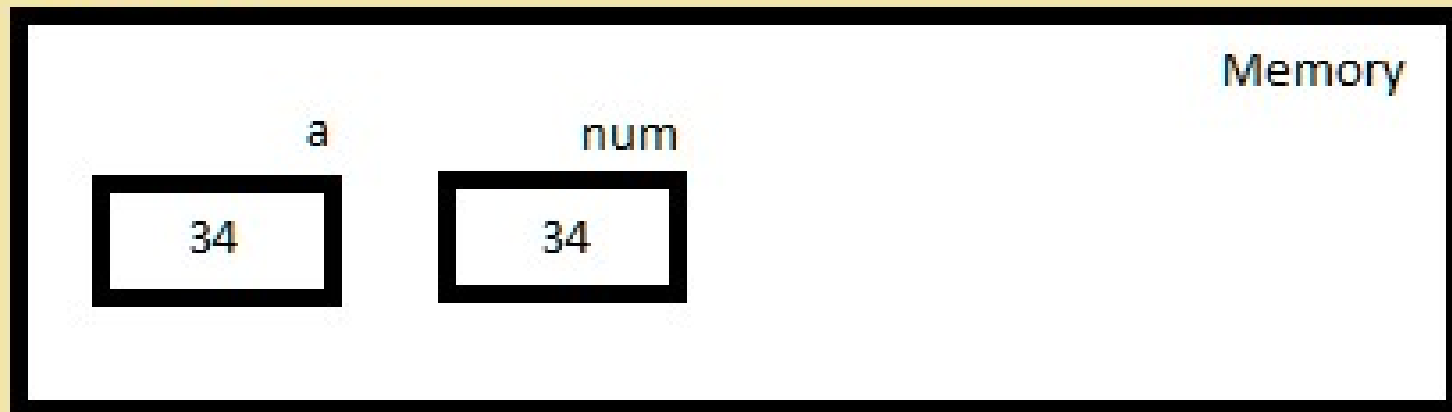
Why?

- Java uses *call-by-value*

- Semantics: when a call is made, the method called works with a **copy** of passed data

Call By Value

public void meth (int num)

int a = 34;
meth (a)

Memory

a   num

34   34

Why?

- Java uses *call-by-value*

- Semantics: when a call is made, the method called works with a **copy** of passed data

```
public static void something(int x) {
   x = 1;
}

public static void
main(String[] args) {
   int x = 8;
   something(x);
   System.out.println(x);
}
```

Why?

- Java uses *call-by-value*

- Semantics: when a call is made, the method called works with a **copy** of passed data

```
public static void something(int x) {
  x = 1;
}

public static void
main(String[] args) {
  int x = 8;
  something(x);
  System.out.println(x);
}
```

something gets a **copy** of x

any changes something makes will only change the **copy**

Testing with JUnit

Testing Motivation

- Builds confidence that code works as intended

- Ensures that code doesn't break if downstream changes are made

JUnit Motivation

- **Wildly** popular for writing tests for Java

- Can do a *lot*

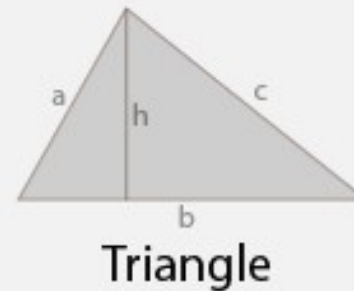# Example: TrianglePerimeter.java

Area A $= \dfrac{bh}{2}$

Perimeter P $= a + b + c$

b $\longrightarrow$ base

h $\longrightarrow$ height

a $\longrightarrow$ side

c $\longrightarrow$ side

Triangle

Key Point 1: Filename

Tests must be held in `MyClassTest.java`, where the code is held in `MyClass.java`

Key Point 1: Filename

Tests must be held in `MyClassTest.java`,
where the code is held in `MyClass.java`

`TrianglePerimeter.java`

Key Point 1: Filename

Tests must be held in `MyClassTest.java`,
where the code is held in `MyClass.java`

```
TrianglePerimeter.java
TrianglePerimeterTest.java
```

Key Point 1: Filename

Tests must be held in `MyClassTest.java`,
where the code is held in `MyClass.java`

`TrianglePerimeter.java`

`TrianglePerimeterTest.java`

`MultiplySeven.java`

Key Point 1: Filename

Tests must be held in `MyClassTest.java`,
where the code is held in `MyClass.java`

`TrianglePerimeter.java`

`TrianglePerimeterTest.java`

`MultiplySeven.java`

`MultiplySevenTest.java`

Key Point 2: `imports`

File containing tests must begin with:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
```

Key Point 3: Method Setup

Each test is a method of the form:

```
@Test

public void testName() {
    ...
}
```

Key Point 3: Method Setup

Each test is a method of the form:

```
@Test

public void testName() {
  ...
}
```

Note: no `static`

Key Point 4:
`assertEquals`

- Test method bodies must contain `assertEquals`, which **fails** the test if the two passed values are **not** equal

- Tests without `assertEquals` **test nothing!**

Key Point 4:
`assertEquals`

- Test method bodies must contain `assertEquals`, which **fails** the test if the two passed values are **not** equal

- Tests without `assertEquals` **test nothing!**

```
@Test public void myTest()  {
    assertEquals(1, 2);
}
```

Key Point 5: ClassName.methodName

To call a method `foo` defined in `Foo.java` from `FooTest.java`, you must say `Foo.foo()`

Key Point 5:
ClassName.methodName

To call a method `foo` defined in `Foo.java` from `FooTest.java`, you must say `Foo.foo()`

```
@Test public void myOtherTest() {
   assertEquals(2, Foo.foo(7));
}
```