

COMP 110/L Lecture 6

Maryam Jalali

Slides adapted from Dr. Kyle Dewey

Outline

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

Variable Scope

Question

Does this compile?

```
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;  
}
```

Same name - does this compile?

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

Yup!

Why?

- Declared variables have a *scope*
- The scope of a variable is the section of code in which a variable is valid or “known.”
- 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 { }

```
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;  
    }  
}
```

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

Scope of main

Same variable
name in same
scope:error

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

Scope of main


```
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;  
}
```

```
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

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

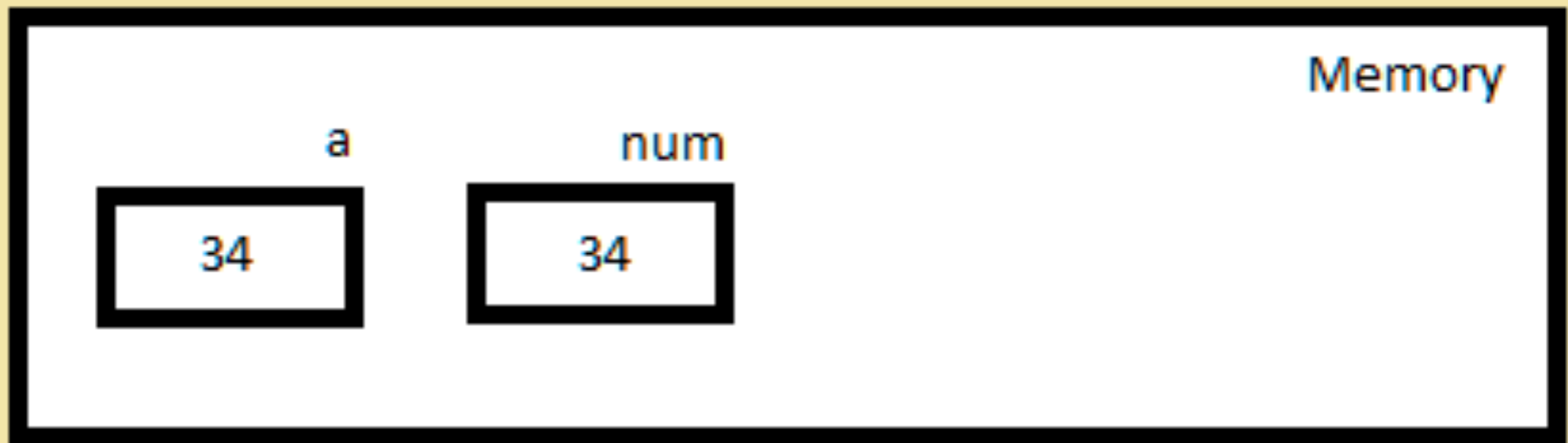
Why?

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

Call By Value

```
int a = 34;  
meth ( a )
```

```
public void meth ( int num )
```



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;  
}
```

something gets a **copy** of *x*

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

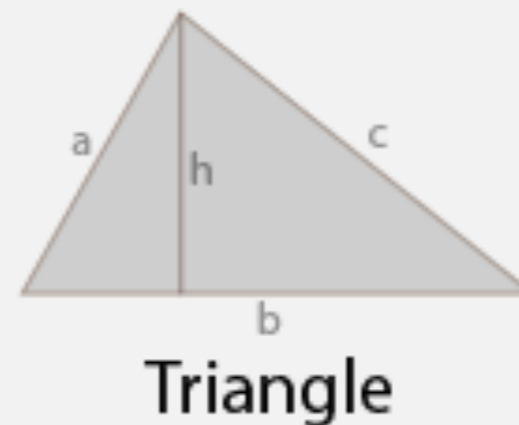
Example:

TrianglePerimeter.java

$$\text{Area } A = \frac{bh}{2}$$

$$\text{Perimeter } P = a + b + c$$

- b → base
- h → height
- a → side
- c → side



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));  
}
```