types and static methods

CSc 115

Celina Berg

Textbook:
pages 4-11, 15, 26, 27

Primitive Data Types

Category	Data Type	Wrapper Class
Boolean	`boolean`	`Boolean`
Character	`char`	`Character`
Integer	`byte`	`Byte`
	`short`	`Short`
	`int`	`Integer`
	`long`	`Long`
Floating point	`float`	`Float`
	`double`	`Double`

Figure 1-5

Primitive data types and corresponding wrapper classes

Variables

- Represents a memory location
- Contains a value of primitive type or a reference
- Its name is a Java identifier
- Declared by preceding variable name with data type

```
double radius; // radius of a sphere
String name; // reference to a String object
```

casting

- Implicit casting
  - Assigning/converting a variable of one type to another of higher precision

  ie. int i = 6;

  double d = i; // after this line: d is 6.0, i is still 6

- Explicit casting
  - Forcing the conversion of a variable to a specified type

  ie. double d = 6.7;

  int i = (int) d; // after this line: i is 6, d is still 6.7

Compound Assignments Operators

Operator	Example	Equivalent example
-=	`a -= 5`	`a = a - 5`
*=	`b += 2`	`b = b + 2`
+=	`c *= 6`	`c = c * 6`
/=	`d /= 3`	`d = d / 3`
%=	`e %= 4`	`e = e % 4`

Other Assignment Operators

Operator	Example	Expanded equivalent version
++	a++	a = a + 1
	++a	a = a + 1
--	a--	a = a - 1
	--a	a = a – 1

++a and a++ appear to have the same affect but not when combined with other operations in one statement…

++a means: a = a + 1 will happen **and** the produced value is what a is AFTER 1 is added to a

a++ means: a = a + 1 will happen **but** the produced value is what a was BEFORE 1 is added to a

```
a = 10;
b = 5 + ++a;
// a is 11, b is 16
```

Expanded version, same result:

```
a = 10;      // a is 10
a = a + 1;   // a is 11
b = 5 + a;   // b is 16
```

```
a = 10;
b = 5 + a++;
// a is 11, b is 15
```

Expanded version, same result:

```
a = 10;     // a is 10
b = 5 + a; // b is 15
a = a + 1; // a is 16
```

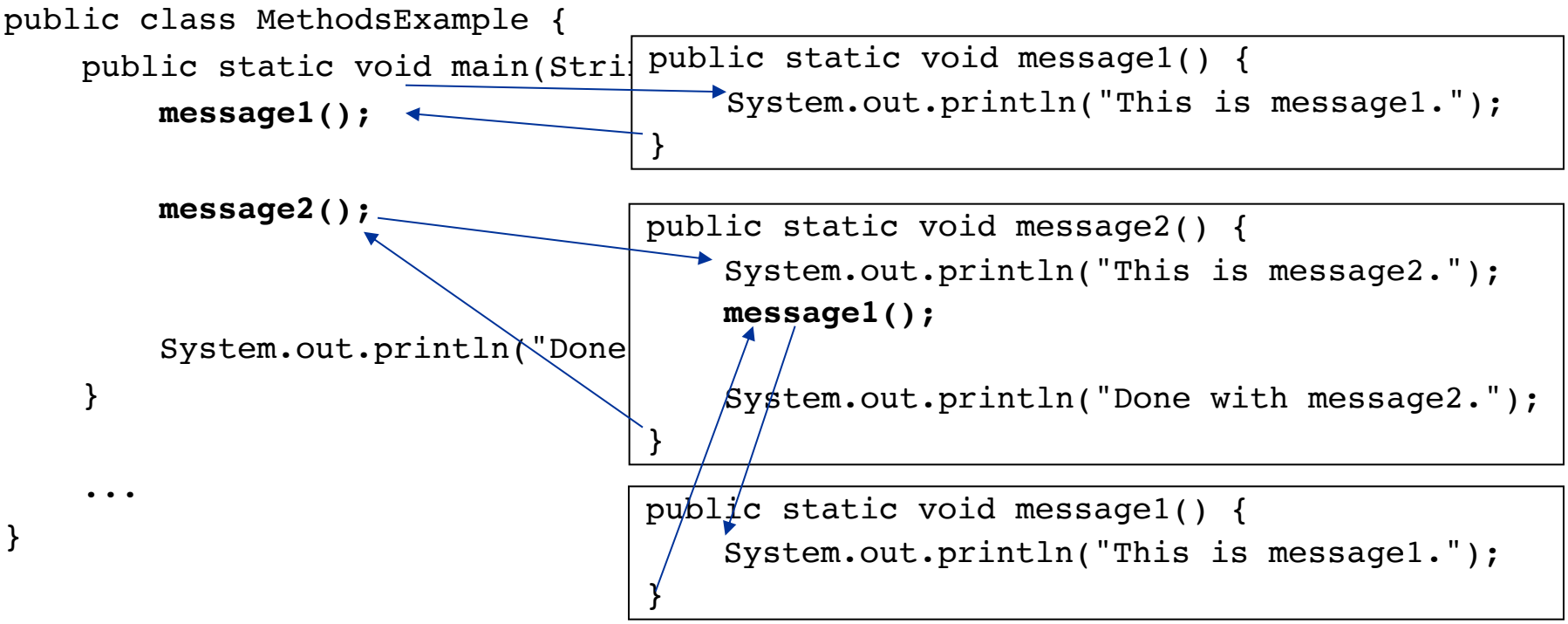
Methods calling methods

```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }

    public static void message1() {
        System.out.println("This is message1.");
    }

    public static void message2() {
        System.out.println("This is message2.");
        message1();
        System.out.println("Done with message2.");
    }
}
```

- What does this program output?

Control flow

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.



Method Parameters

- We can also give our methods input as parameters, this is called passing in an **arguement**.

```
public static void name (type name)
{
    ..code here..
}
```

type examples: **int**, **double**, **String**

```
public static void addFive (int num1) {…}
```

- To call this method:

```
addFive(6);
```

```
addFive(6.5); why won`t this work?
```

Return values

- When we create the method, we designate the type of variable the method will return

```
public static int name (type name) {
    ...
}
```

Put `int`, `double`, `String`, etc here

`void` means we don't return anything.

```
public static int addFive (int num) {
  int newNumber = num+5;
  return newNumber;
}
```

- To call this method:

```
int result = addFive(3); //result is now 8
```