JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. Classes and Instances
2. **The Methods**
3. The Constructors
4. Static Elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and Interfaces
9. String processing
10. Wrapper classes for primitive types
11. Exceptions and Assertions
12. Nested classes
13. Enums
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java classes
19. Object Oriented Design
20. Functional Programming

Module contents

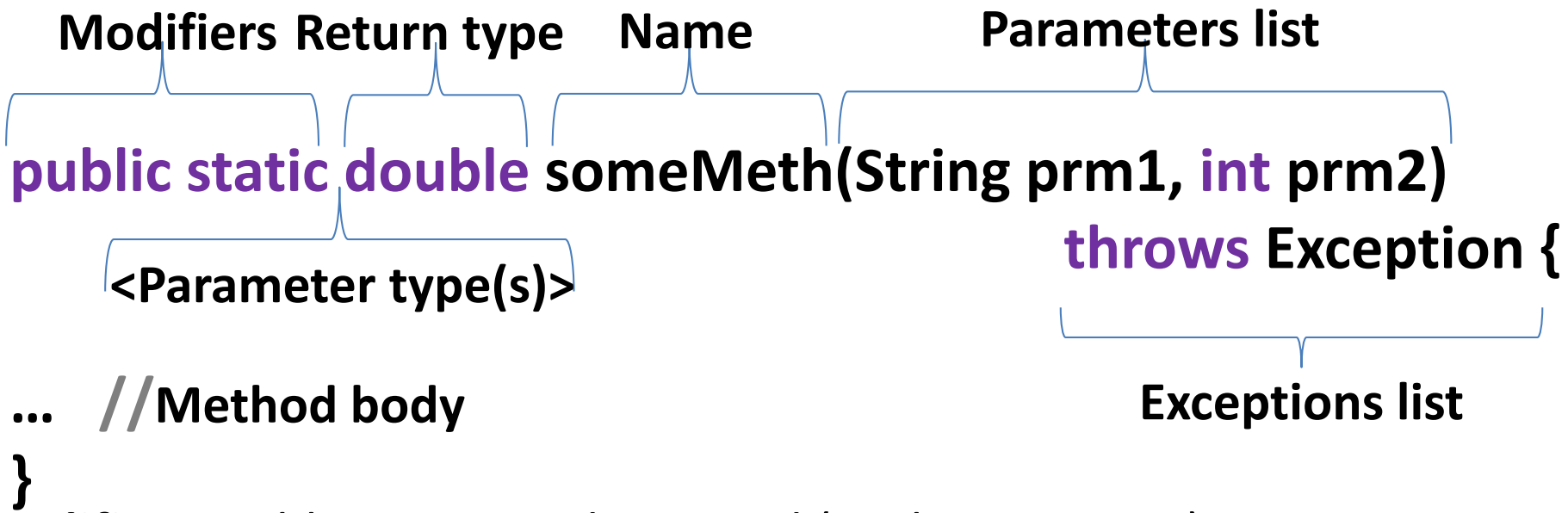
- The Methods
  - Methods declaration
  - Passing arguments to a method
  - Methods with variable arguments
  - Overloading methods

Module contents

- The Methods
  - Methods declaration
  - Passing arguments to a method
  - Methods with variable arguments
  - Overloading methods

Methods declaration 1/3

- A *method* in Java is a block of statements that has a name and can be executed by calling (*invoking)* it from some other place in your program

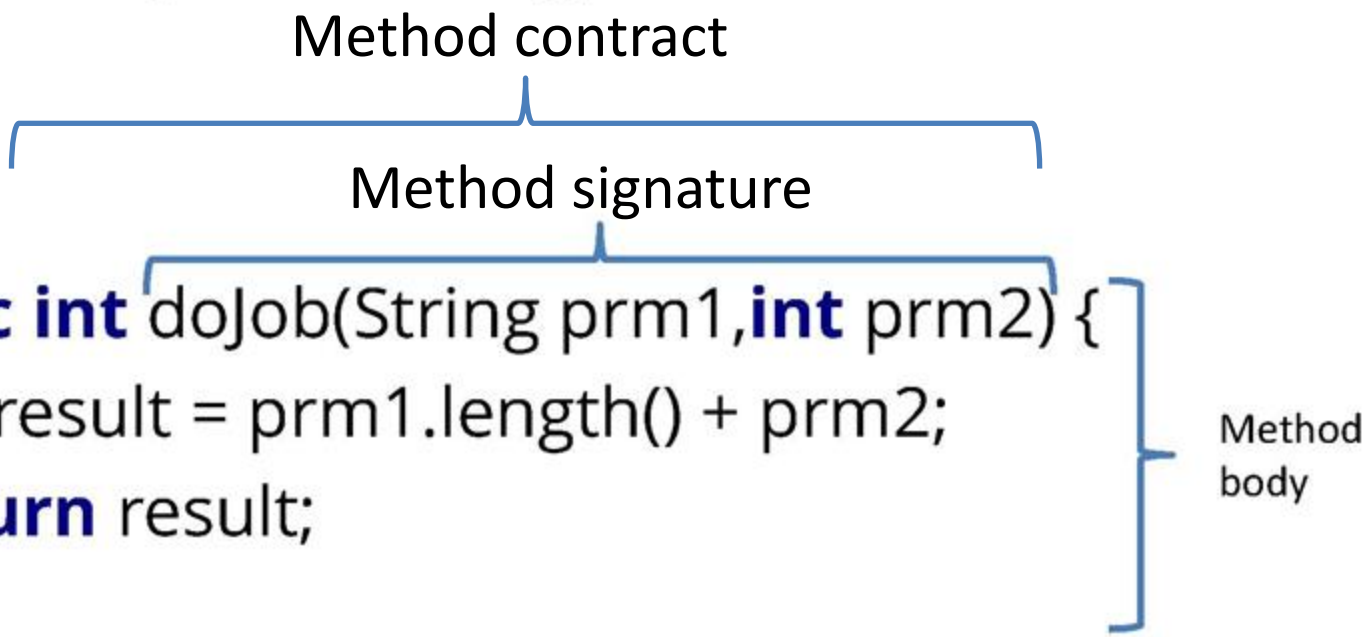


Modifiers: public, protected, omitted (package-private) private, static, final, abstract, synchronized, native

Return type: primitive, reference, void

Methods declaration 2/3

- Two of the components of a method declaration comprise the *method signature*—the method's name and the parameter types



Method contract = Signature + Return type

Methods declaration 3/3

```
1.  public class Car {
2.      //...
3.      private int speed;
4.      //...
5.      public int getSpeed() {
6.          return speed;
7.      }
8.      public void setSpeed(int speed) {
9.          this.speed = speed;
10.     }
11. }
```

starts with verb

End of the method: complete all operators OR return OR Exception

Module contents

- The Methods
  - Methods declaration
  - Passing arguments to a method
  - Methods with variable arguments
  - Overloading methods

Passing arguments to a method 1/4

- *Parameters* refers to the list of variables in a method declaration.
- *Arguments* are the actual values that are passed in when the method is invoked. When you invoke a method, the arguments used must match the declaration's parameters in type and order.

PassPrimTypeToVoidMeth

PassRefTypeToVoidMeth

PassParameterWithCopy

Passing arguments to a method 2/4

pass by reference

```
cup =
```

```
fillCup(      )
```

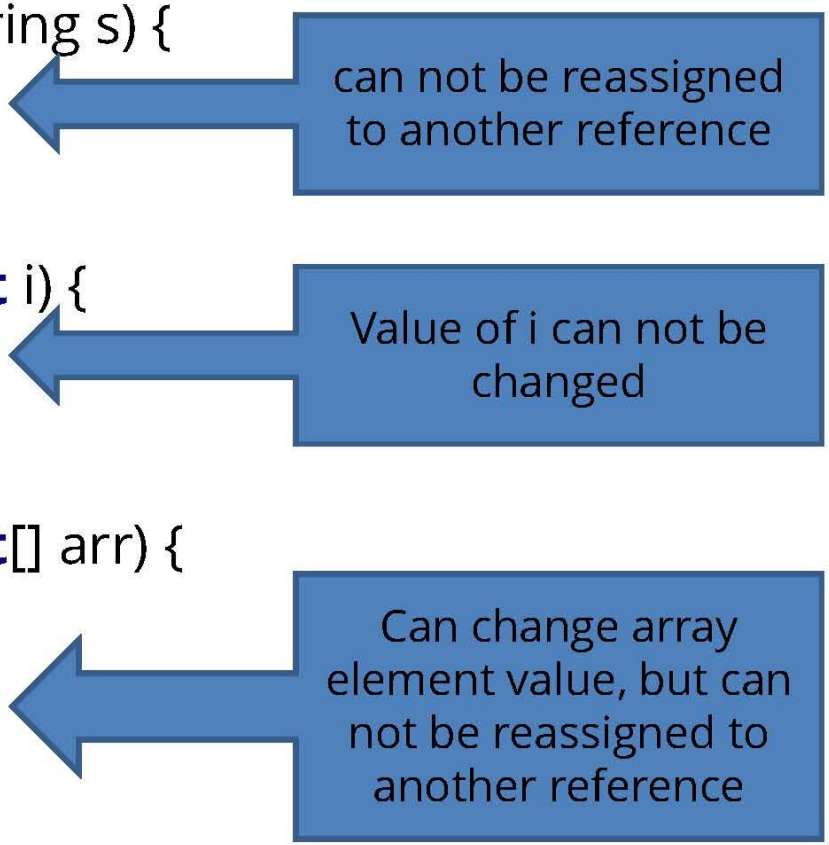
*pass by **value***

```
cup =
```

```
fillCup(      )
```

Final modifier for method arguments

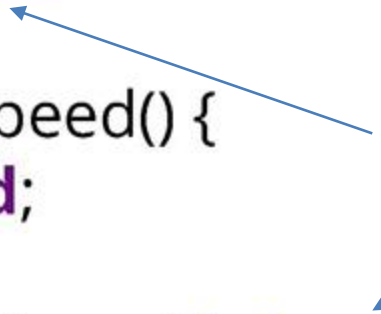
```
1.  public void test(final String s) {
2.    s = "abcd";
3.    // ...
4.  }
5.  public void test(final int i) {
6.    i++;
7.    // ...
8.  }
9.  public void test(final int[] arr) {
10.   arr[0] = 100;
11.   arr = new int[10];
12.   // ...
13. }
```



See finalargpkg

Class field shadowing by method parameter

```
1.  public class Car {
2.      //...
3.      public int speed;
4.      //...
5.      public int getSpeed() {
6.          return speed;
7.      }
8.      public void setSpeed(int speed) {
9.          this.speed = speed;
10.     }
11. }
```



class field shadowing

Return from method

```
public class MyClass {
    public int meth1(){
        return 0;
    }
    public int meth2(){
        return (byte)0;
    }
    public String meth3() {
        return null;
    }
    public void meth4(){
        return;
    }
}
```

void methods may contain return statement

The return statement need not be the *last statement* in a method, but it must be the *last statement to execute* in a method

Module contents

- The Methods
  - Methods declaration
  - Passing arguments to a method
  - Methods with variable arguments
  - Overloading methods

See varargspkg

since Java 5

Module contents

- The Methods
  - Methods declaration
  - Passing arguments to a method
  - Methods with variable arguments
  - Overloading methods

Instance or STATIC methods or constructors can be defined as overloaded

See methoverloading