Methods in Java

Josh Wilcox (jw14g24@soton.ac.uk)

January 29, 2025

Contents

1 How to define a Method

- Determine whether it is public / private
- Determine whether it is static or not
- **Always** determine what type it returns
  - Can be primitive or a class, or *void* if there is nothing to return
- Give the method name
  - Should start with lower case letters
- Give the parameters if required

2 Primitives VS Objects as Parameters

```
1  public class Zoo{
2      public static void main(String[] args){
3          int visitors = 0;
4          Elephant elephant = new Elephant();
5          incVisitors(visitors);
6          feedElephant(elephant);
7          System.out.println(visitors);
8          System.out.println(elephant.isHungry());
9      }
10     public static void incVisitors(int v) {
11         if(zooIsOpen()) v++;
12     }
13     public static void feedElephant(Elephant e) {
14         if(zooHasFood()) e.feed();
15     }
16
17     //some code omitted
18 }
```

- The `visitors` variable is a primitive type (`int`). When passed to `incVisitors`, a copy of its value is made, so changes to `v` inside the method do not affect the original `visitors` variable.
- The `elephant` variable is an object of the `Elephant` class. When passed to `feedElephant`, a reference to the same object is passed, so changes to the `elephant` inside the method affect the original `elephant` object.
- After calling `incVisitors(visitors)`, the value of `visitors` remains unchanged because primitives are passed by value.
- After calling `feedElephant(elephant)`, the state of the `elephant` object may change (e.g., it is no longer hungry) because objects are passed by reference.

3 Overloading

- Methods are recognized by their **signatures**, which include:
  - Method name
  - Number and type of parameters
  - Order of parameters

- When overloading a method, each overloaded method must have a unique signature.
  - The return type is **not** part of the method signature and cannot be used to distinguish overloaded methods.
- Overloading allows methods to perform similar functions with different inputs.
- Example of method overloading:

```java
public class MathUtils {
    // Overloaded method for adding two integers
    public static int add(int a, int b) {
        return a + b;
    }

    // Overloaded method for adding three integers
    public static int add(int a, int b, int c) {
        return a + b + c;
    }

    // Overloaded method for adding two double values
    public static double add(double a, double b) {
        return a + b;
    }
}
```

- In the example above, the `add` method is overloaded with different parameter lists:
  - `add(int a, int b)`: Adds two integers.
  - `add(int a, int b, int c)`: Adds three integers.
  - `add(double a, double b)`: Adds two double values.
- The appropriate method is called based on the arguments passed during the method call.