Programming in Java – Day 4 Recap

Methods

School of Computing and Mathematical Sciences
Birkbeck, University of London

Topics on Day 4

- Methods
- Formal and actual parameters
- Calling methods
- Scope
- Flow of execution
- Methods in classes: instance methods
- `for` loops

Don't Repeat Yourself!

Overarching principle in programming:

Don't **R**epeat **Y**ourself!

Methods

- reusable (!) mini-programs with a name, some inputs, and a result

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword `static`

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

- Usage (elsewhere) by **method call**:

```
1        int a = 5;
2        int sum = add(a, 10);
```

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

- Usage (elsewhere) by **method call**:

```
1        int a = 5;
2        int sum = add(a, 10);
```

- **Formal parameters** of add:

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

- Usage (elsewhere) by **method call**:

```
1        int a = 5;
2        int sum = add(a, 10);
```

- **Formal parameters** of add: `int` op1, `int` op2

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

- Usage (elsewhere) by **method call**:

```
1        int a = 5;
2        int sum = add(a, 10);
```

- **Formal parameters** of add: `int` op1, `int` op2
- **Actual parameters** (or **arguments**) of the call to add:

Methods

- reusable (!) mini-programs with a name, some inputs, and a result
- inputs and result have **types**
- "standalone method": keyword **`static`**
- Example:

```
1    static int add(int op1, int op2) {
2        int result = op1 + op2;
3        return result; // gives the result back
4    }
```

- Usage (elsewhere) by **method call**:

```
1        int a = 5;
2        int sum = add(a, 10);
```

- **Formal parameters** of add: `int op1, int op2`
- **Actual parameters** (or **arguments**) of the call to add: `a, 10`

Scope: where can we use our variables?

```
1  public class Example {
2      static void myMethod(int a) {
3          System.out.println(a);
4          int b = 42;
5          while (b > 0) {
6              int c = 3;
7              b -= c;
8          }
9          System.out.println("Done looping!");
10     }
11     public static void main(String[] args) {
12         myMethod(8);
13     }
14 }
```

What are the **scopes** of a, b, c, args?

Method parameters with complex types

```
1  class Point {
2      int x; // no y in this example
3  }
```

```
1  public class MethodCallsCopyBoxesOnTheStackButNothingOnTheHeap {
2      static void increment(Point point, int n) {
3          n = n + 1;
4          point.x = point.x + 1;
5          point = null;
6      }
7      public static void main(String[] args) {
8          Point myPoint = new Point();
9          myPoint.x = 0;
10         int myInt = 0;
11         increment(myPoint, myInt);
12         System.out.println("The integer is now " + myInt);
13         System.out.println("The point is now " + myPoint.x);
14     }
15 }
```

Flow of execution

```
1  public class D4Example3 {
2      static int add(int op1, int op2) {
3          int result = op1 + op2;
4          return result;
5      }
6      static void doSomething() {
7          int a = 5;
8          int sum = add(a, 10);
9          System.out.println(sum);
10     }
11     public static void main(String[] args) {
12         doSomething();
13     }
14 }
```

What happens on the stack?

Instance methods

```
1  class UnidimensionalPoint {
2      int x; // instance variable
3
4      int getX() {
5          return x;     // x is from line 2
6      }
7
8      void setX(int x) {
9          this.x = x;  // this.x is from line 2
10     }                //      x is from line 8 (!)
11
12     UnidimensionalPoint clone() { // no static!
13         UnidimensionalPoint copy = new UnidimensionalPoint();
14         copy.setX(x); // method call: varName.methodName(arg)
15         return copy;
16     }
17 }
```

for loops

```
1           for (int i = 0; i < 42; i++) {
2               System.out.println(i);
3           }
4           System.out.println("Done looping!");
```

for loops

```
1        for (int i = 0; i < 42; i++) {
2            System.out.println(i);
3        }
4        System.out.println("Done looping!");
```

vs

```
1        int i = 0;
2        while (i < 42) {
3            System.out.println(i);
4            i++;
5        }
6        System.out.println("Done looping!");
```

for loops

```
1        for (int i = 0; i < 42; i++) {
2            System.out.println(i);
3        }
4        System.out.println("Done looping!");
```

vs

```
1        int i = 0;
2        while (i < 42) {
3            System.out.println(i);
4            i++;
5        }
6        System.out.println("Done looping!");
```

What is the scope of `i` in each snippet?

Topics on Day 4

- Methods
- Formal and actual parameters
- Calling methods
- Scope
- Flow of execution
- Methods in classes: instance methods
- `for` loops