Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Operations methods and operation capability

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Op(eration) methods – *JR book chapter 3*

- JR Op-method declaration == Java method declaration
- Keyword: `op`
- Invocation:
  - Normal Java method
  - Invoked via `call` statement

Op methods

```
import edu.ucdavis.jr.JR;

public class Basic {
    private static op int square(int x) {
        System.out.println("in square "+x);
        return x*x;
    }
    public static void main(String [] args) {
    System.out.println(square(23));
        square(41);
        call square(41);
    }
}
```

Op methods

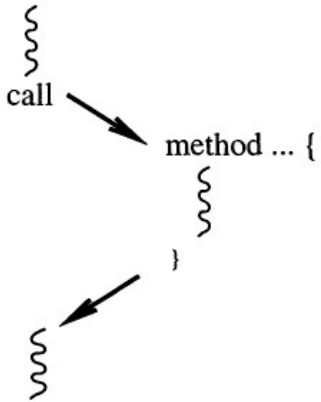
An op-method declaration is really an **abbreviation** for an operation declaration and an ordinary Java method.

```
import edu.ucdavis.jr.JR;
public class Basic {
     private static op int square(int);
     private static int square(int x) {
          System.out.println("in square "+x); return x*x;
     }
     public static void main(String [] args) {
          System.out.println(square(23));
          square(41);
          call square(41);
     }
}
```

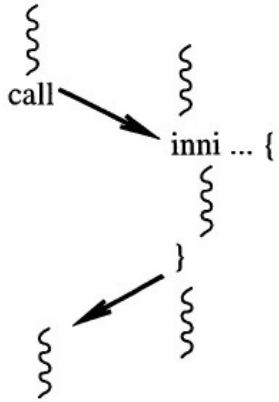
Operations, invocation and service

- Invocations to ops can be
  - synchronous (`call`)
  - asynchronous (`send`)
- Serviced by
  - a method (`op`)
  - input statement (`inni`) →

Synchronous



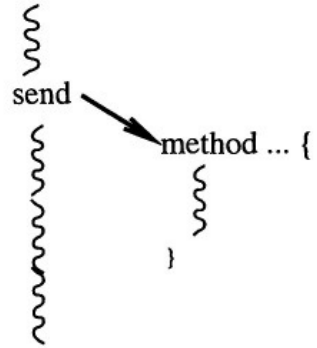
remote procedure call



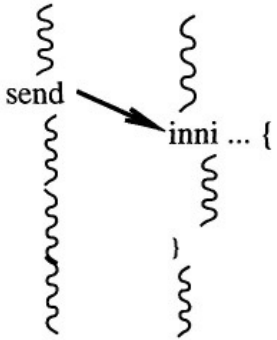
rendezvous

Serviced by: **method (`op`)** **input statement (`inni`)**

Asynchronous



dynamic process creation



asynchronous message passing

Serviced by: **method (`op`)** **input statement (`inni`)**

Operation capabilities

- An operation capability **is a pointer** to (or reference to) an **operation**.
  - *Java offers no function pointers or references*

- Such pointers can be assigned to variables, passed as parameters, and used in invocation statements

- invoking a capability has the effect of invoking the operation to which it points.

Operation capabilities

- A variable or parameter is defined to be an operation capability by declaring its type in the following way:

```
cap operation_specification cap_id
```

- When parameterization is compared, only the **signatures** of formals and return values matter.
  - formal and return **identifiers** are ignored

Op Caps – Comparing

- Capabilities can also be compared using the `==` and `!=` relational operators
- Other relational operators (e.g., <) are not allowed

- Capability variables can also take on two special values:
  - `null` → invocation of `null` cap causes a run-time error
  - `noop` → invocation of `noop` cap has no effect

Ops and caps – Example

```
import edu.ucdavis.jr.JR;

public class Simple {
    // declare a few operations
    private static op void d(int);
    private static op int e(int);
    private static op double f(double);
    private static op double g(double);

    // declare a few capabilities
    private static cap void (int) x, z;
    private static cap double (double) y;

    private static void d(int x) {
       System.out.println("d "+x);
    }
    private static int e(int x) { return -x;}
    private static double f(double x){return x*10; }
    private static double g(double x){return 10000-x;}

    public static void main(String [] args) {
        x = d; // x now points to operation d
        x(387); // invoke operation d with argument 387
        // make y point to one of f or g
        if (e(9) > 0) { y = f; }
        else          { y = g; }
        // invoke what y points to
        System.out.println(y(4.351));
        // capabilities can be assigned and compared
        z = x;
        if (y == f) { System.out.println("y is f"); }
        else        { System.out.println("y is g"); }
     }
    }
```

Ops and caps – Other example

```
public class OpsCaps {
    // return area under the curve f(x) for a <= x <= b
    // using trapezoidal rule with n intervals
    public static op double trapezoidal(double a, double b, int n,
                                        cap double (double) f) {
        double area = 0;
        double x = a;
        double h = (b-a)/n;
        area = (f(a)+f(b))/2;
        for (int i = 1; i <= n-1; i++) {
            x += h; area += f(x);
        }
        area *= h;
        return area;
    }

    // a spooky function
    private static op double fun1(double x) {
        return x*x + 2*x +4;
    }

    // right triangle
    private static op double fun2(double x) {
        return x;
    }

    // sinus wave
    private static op double fun3(double x) {
        return Math.sin(x);
    }
```

```
    public static void main(String [] args) {
        double area;    // calculated area under the fuction

        int start, end, interval; // arguments that will be hardcoded

        cap double (double) f;

        // hardcode some values
        start = 0;
        end = 10;
        interval = 500;

        f = fun1;
        area = trapezoidal(start, end, interval, f);
```

Caps – Exercises

```
private static cap int (char) z;

private static cap void () a;

private static cap double (int) b;

private static cap int (char, double) c;

private static cap int (cap double (char)) d;

private static cap cap void () (int) e;

private static cap cap int (boolean) (cap double (char)) f;
```