Classes as the basic component

- Classes have a dual role:
  - They are software modules
  - They are data types
- Methods define the behaviour of objects of a class.

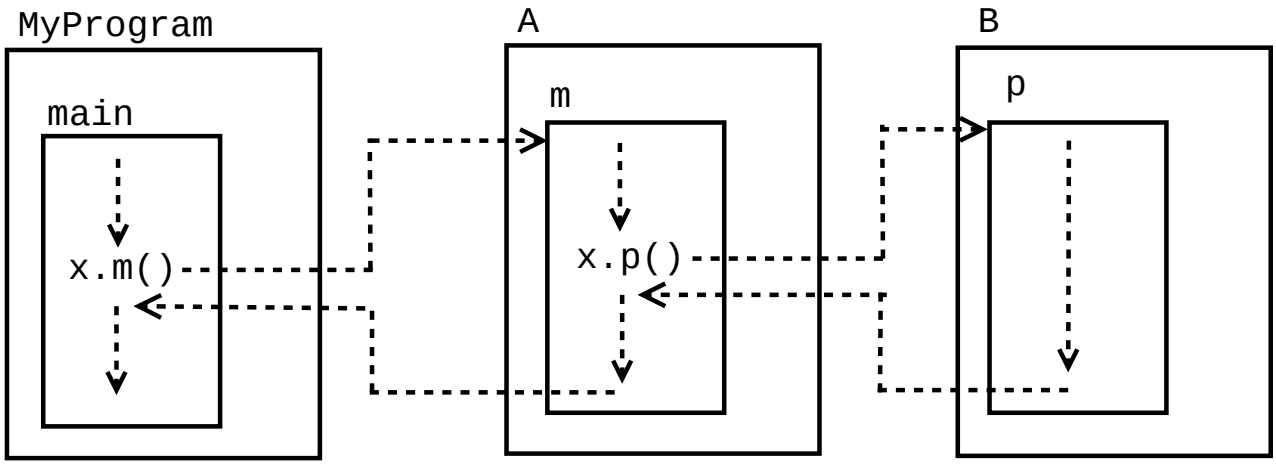
Program Structure

```
public class MyProgram {
  public static void main(String[] args)
  {
    //...
  }
}

public class A {
  //...
}

public class B {
  //...
}
```

Method invocation: control flow

```
public class MyProgram {
  public static void main(String[] args)
  {
    A x = new A();
    x.m();
  }
}
public class A {
  void m()
  {
    B x = new B();
    x.p();
  }
}
public class B {
  void p()
  {
    System.out.println(“Do something”);
  }
}
```

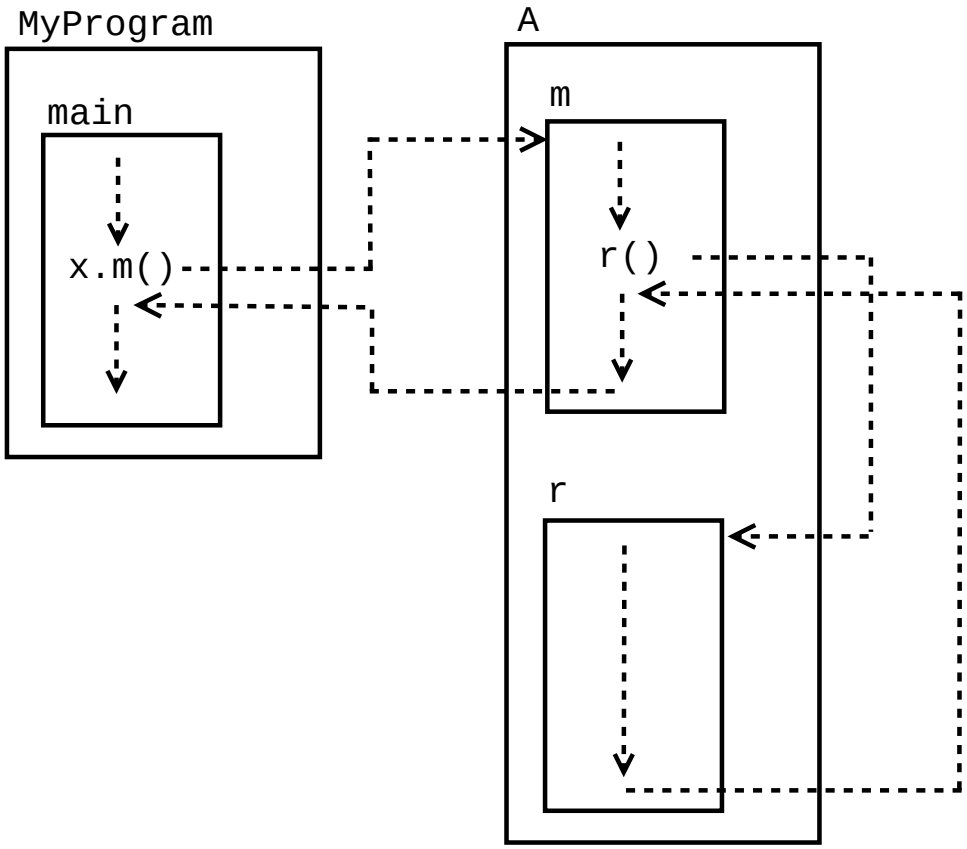
Method invocation: control flow



Method invocation: control flow; "this"

```
public class MyProgram {
  public static void main(String[] args)
  {
    A x = new A();
    x.m();
  }
}
public class A {
  void m()
  {
    r();    // Equivalent to this.r();
  }
  void r()
  {
    System.out.println("Do something");
  }
}
```

Method invocation: control flow



Method invocation: parameter passing

- A *frame* is a space in memory which stores a set of variables. It can be viewed as a table containing the memory locations for each variable in the set.

- Suppose that a method is declared as follows:

```
type method(type1 param1, type2 param2,
            ..., typen paramn)
{
    statements;
}
```

- A method call of the form

```
variable.method(arg1, arg2, ..., argn)
```

...where *arg1*, *arg2*, ..., *argn* are expressions with type matching the types as appear in the method declaration, is executed by

First: evaluating each of the arguments *arg1*, *arg2*, ..., *argn* from left to right,

Second: creating a *frame*, reserving space for all the parameters of the method, and local variables declared in the body of the method. The frame also contains a pointer to the object refered to by the *variable*.

Third: in that frame, perform the assignments *param1* = *arg1*; *param2* = *arg2*; ...; *paramn* = *argn*;

Fourth: "jumping" to the body of the method and executing the *statements* in order. The calling method is suspended while the called method is executed.

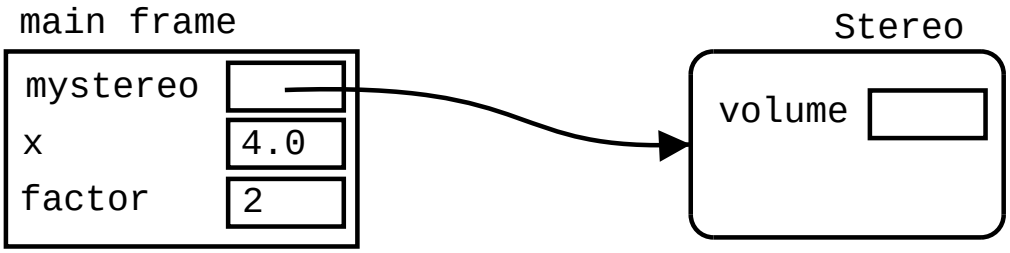
Fifth: when the end of the method is reached, or a `return` statement is reached, stop the method, the frame is discarded, and return to the calling method. The calling method is then resumed in the instruction immediatly after the method call.

Method invocation: Example

```
public class Stereo {
    double volume;
    void set_volume(double v)
    {
        volume = v;
    }
    double get_volume()
    {
        return volume;
    }
}
public class SoundSystem {
    public static void main(String[] args)
    {
        Stereo mystereo = new Stereo();
        double x, factor = 2;
        System.out.println("Testing...");
        x = 4.0;
        mystereo.set_volume(x*factor);
        System.out.println(mystereo.get_volume());
    }
}
```

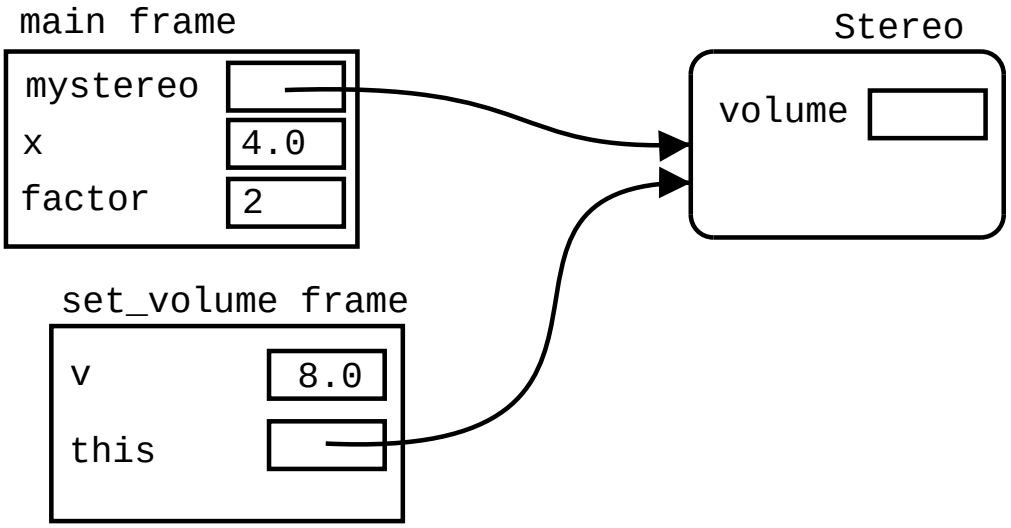
Mehtod invocation: Memory structure

Before calling `mystereo.set_volume(x*factor)`



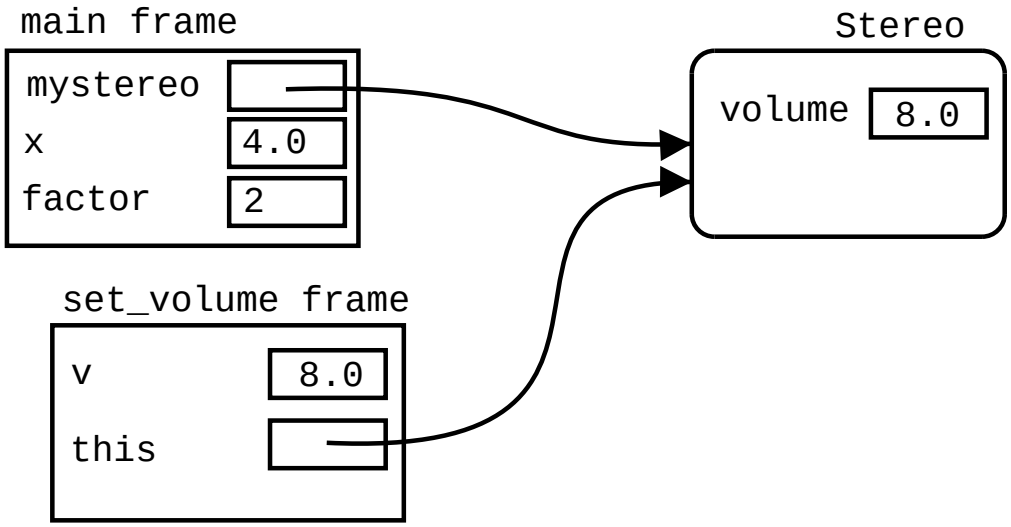
First its arguments (`x*factor`) are evaluated:
Evaluating `x*factor` in the main frame results in `8.0`

A frame for set_volume is created, and
the argument is assigned to the parameter: `v = 8.0;`

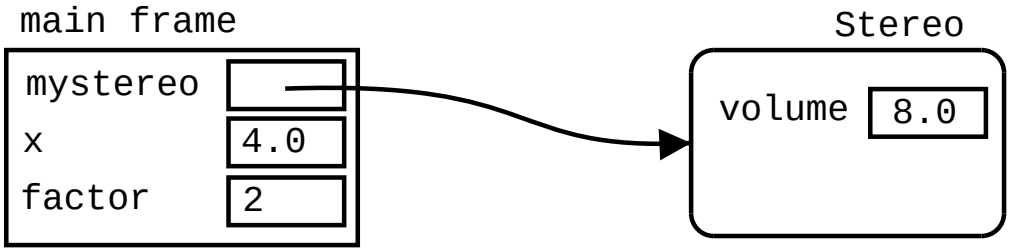


Mehtod invocation: Memory structure

The current method (main) is suspended, and
the body of the called method (set_volume) is executed
in the context of the current frame (the set_volume frame):



Finally the called method frame is discarded, and
computation of the calling method (main) is resumed
in the instruction immediately after the method call.



Scope

- Attributes of a class are shared between its methods:

```
public class F
{
    int n;
    void p()
    {
        n = 3;
        //...
    }
    boolean q(String s)
    {
        if (n < 5 && s.equals("hello"))
            return true;
        return false;
    }
}
```

Scope (contd.)

- ...but are different for different objects of the same class:

```
public class H
{
    void w()
    {
        F f1, f2;
        f1 = new F();
        f2 = new F();
        f1.p();
        f2.p();
        boolean a,b;
        a = f1.q(“hello”);
        b = f2.q(“good bye”);
    }
}
```

- In this example, f1.n and f2.n are different variables of the same class, because they belong to different objects of class F.

Objects are "first class citizens"

- Since classes are data types and objects are their values, then we can do with objects the "same" things that we can do with primitive values, namely:
  - We can assign objects to variables,
  - We can pass objects as arguments to methods, and
  - Methods can return objects as their result.

Objects are "first class citizens" (contd.)

- Variables, attributes can be declared as having a class for its type:

```
Stereo mystereo, yourstereo;
```

- Variables whose type is a class can be assigned objects of that class:

```
mystereo = new Stereo();
yourstereo = mystereo;
```

- Objects can be passed as parameters; if there is a method `void m(Stereo s) {...}` in some class `C`, then we can do:

```
C x = new C();
x.m(mystereo);
x.m(yourstereo);
x.m(new Stereo());
```

Objects are "first class citizens" (contd.)

- Objects can be returned as values; if there is a method

```
Stereo p()
{
    return new Stereo();
}
```

in some class C, then we can do:

```
C x = new C();
mystereo = x.p();
```

...provided that the variable which is being assigned is of the same type.

Example

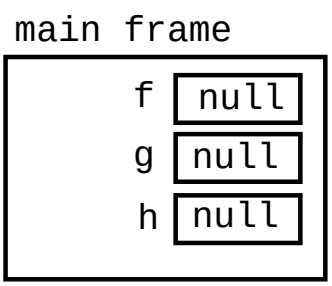
```
public class A {
    int k;
    A()         // Constructor
    {
        k = 1;
    }
}
public class B {
    A x;          // Objects can be attributes;
    void m()
    {
        x = new A();
    }
    void p(A u) // Parameters may have a class
    {           //    for type
        x = u;  // The object u is created
    }           //    elsewhere
    A r()
    {
        return x;
    }
}
```

Example (contd.)

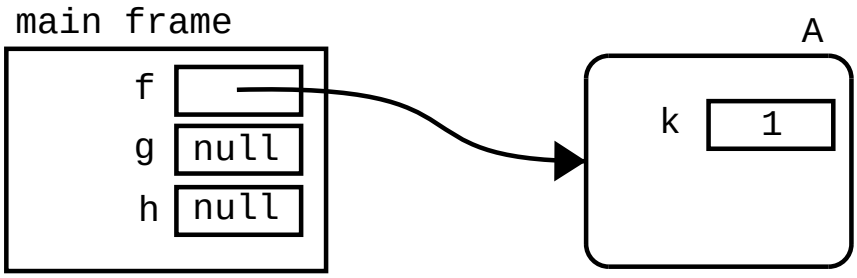
```
public class C {
  public static void main(String[] args)
  {
    A f,g;  // f and g are initialized to null
    B h;    // h is initialized to null
    f = new A();
    // Here f.k is 1
    f.k = 5;
    h = new B();
    h.m();    // assigns a new A to h.x, so ...
    // Here h.x.k is 1
    h.p(f);   // object f is passed as argument
    // Here h.x is f, and therefore h.x.k is 5
    // Also, g is still null, so there is no g.k
    g = h.r();
    // Now g is the same as h.x, which is f,
    // ...so g.k is 5
  }
}
```

Example (contd.)

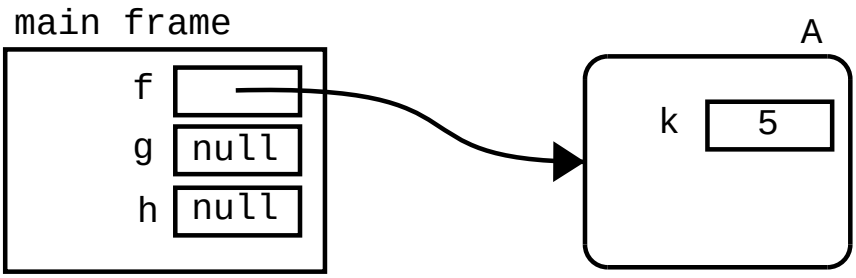
The variables are initialized to null



f is assigned a new A object

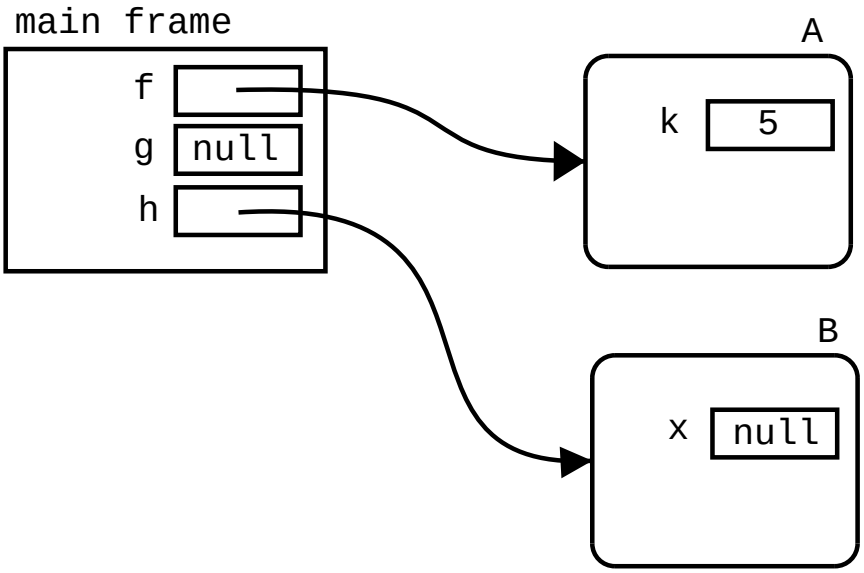


The statement `f.k = 5;` is executed

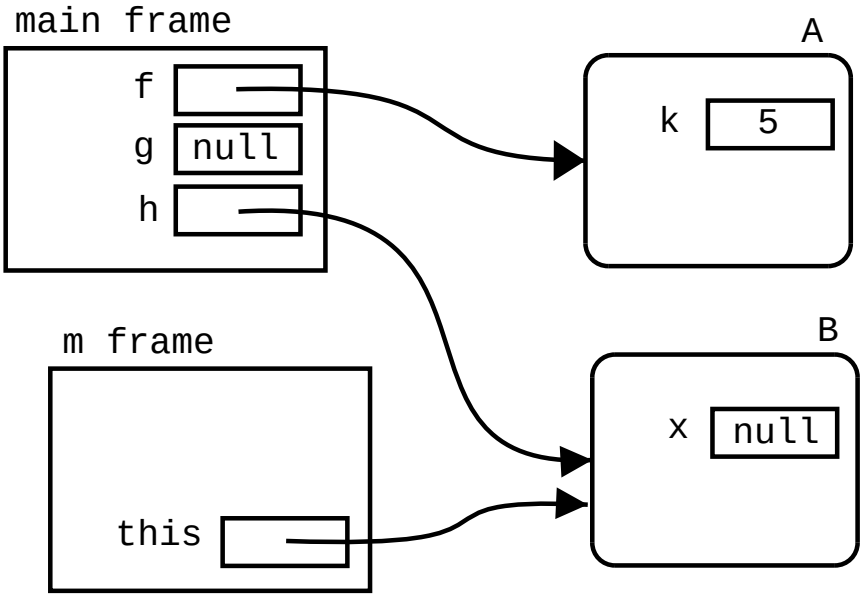


Example (contd.)

h is assigned a new B



We call `h.m()` which creates a frame for m with no arguments

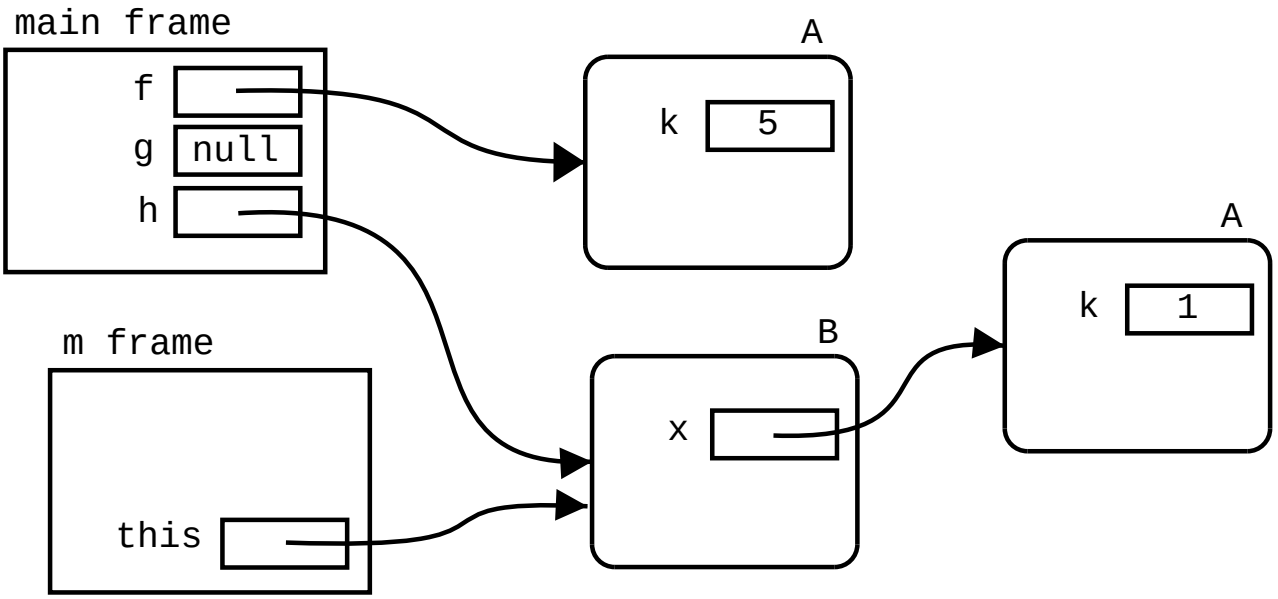


Example (contd.)

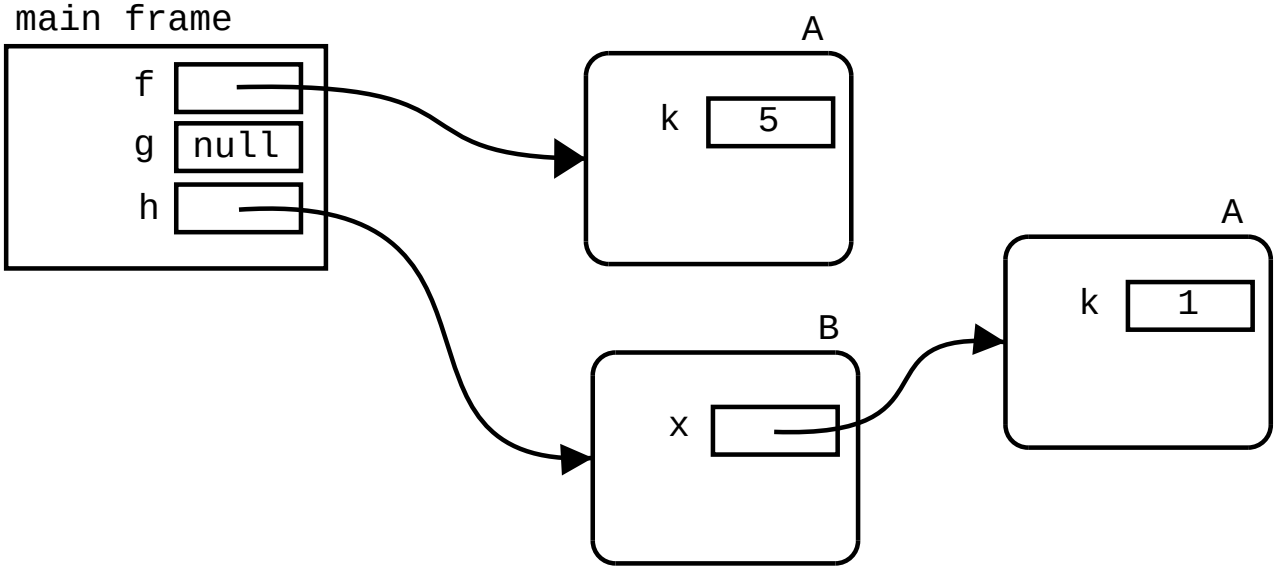
The body of m is executed. It consists of the single statement

```
x = new A();
```

which creates a new A object and assigns it to `this.x`

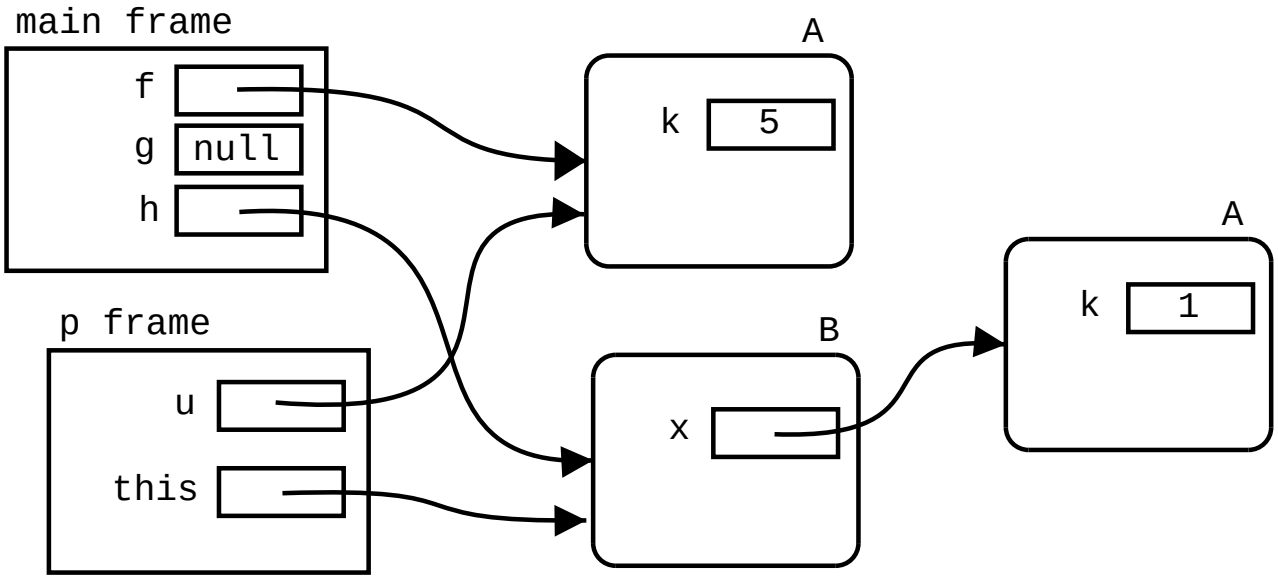


After returning from m, its frame gets discarded, and `h.x.k` is `1`

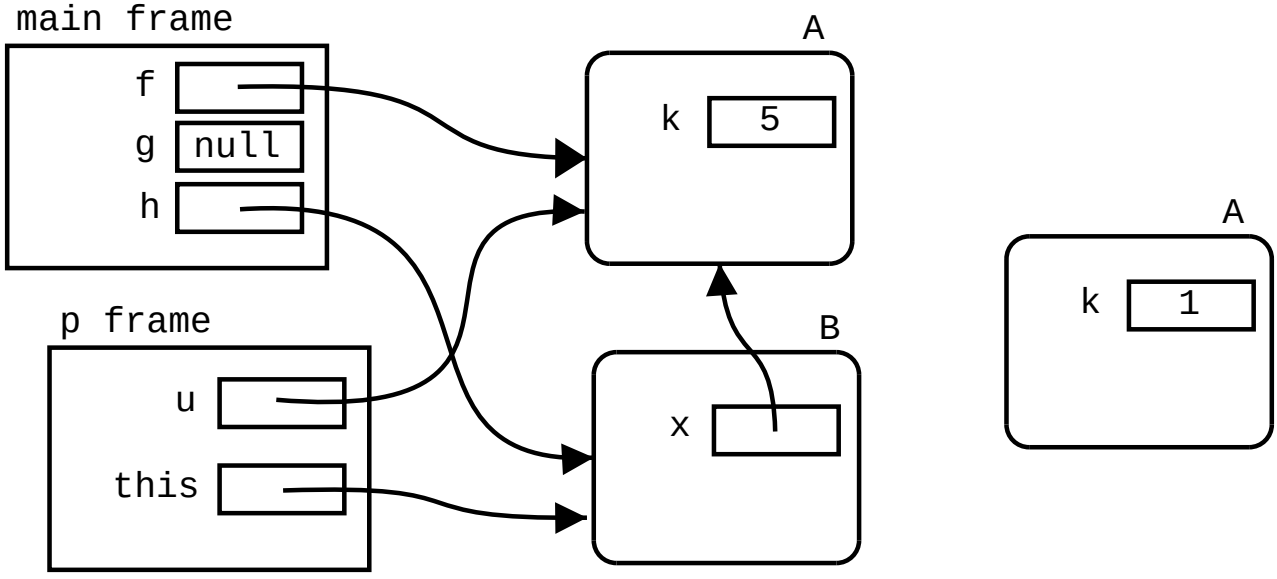


Example (contd.)

Computation in main continues with `h.p(f);`
A frame for p is created, assigning f to its parameter u

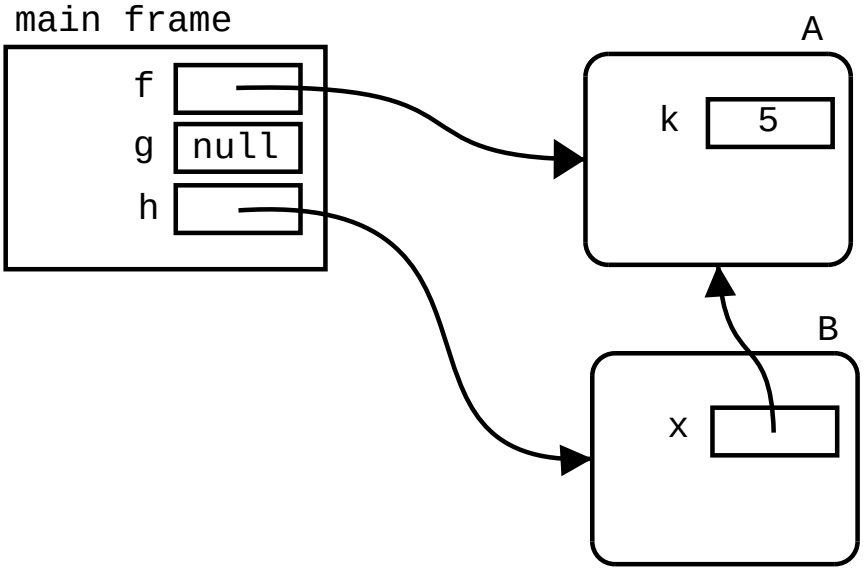


In this frame, the body of p is executed.
The body of p is `x = u;` which is the same as `this.x = u;`

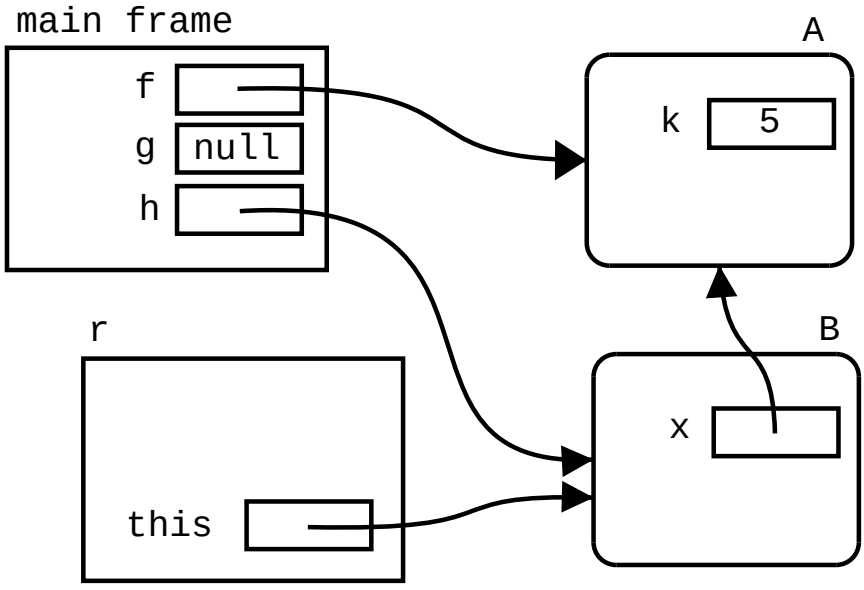


Example(contd.)

When p ends, its frame is discarded.
The other A object that has no references to it, is also discarded.



Computation is resumed with the next instruction of the main: `g = h.r();`
So the right-hand side of the assignment, `h.r()` is evaluated. So a frame for r is created.



Example (contd.)

The body of r is executed in this frame. Its body is `return x;` which is the same as `return this.x;` But `this.x` is the same as the pointer to f, so this pointer is returned, discarding the frame for r, and performing the pending assignment to g, which is now equivalent to: `g = h.x;` or `g = f;`