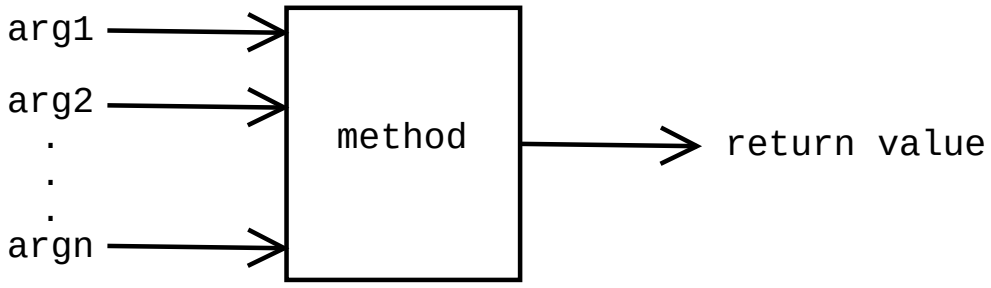
# Methods as functions

- Methods can be viewed as a "black box" with inputs and outputs:




- There are three kinds of methods:
  - Mutators: Modify the state of objects,
  - Accessors: Return information about the object,
  - Constructors: Initialize a newly created object.

# Constructors

- Special methods, whose syntax is given by

```
class_name(list_of_arguments)
{
    statements;
}
```

- For example:

```
public class Student {
    //...
    Student(String n, long i)
    {
        name = n;
        id = i;
    }
    //...
}
```

# Constructors (contd.)

- A constructor method gets executed when a new object of the class gets created using the new keyword. Therefore, the general syntax for the expression used to create objects is:

```
new class_name(list_of_actual_arguments);
```

- For example

```
Student al;
al = new Student(“Alan Turing”, 110011223331);
```

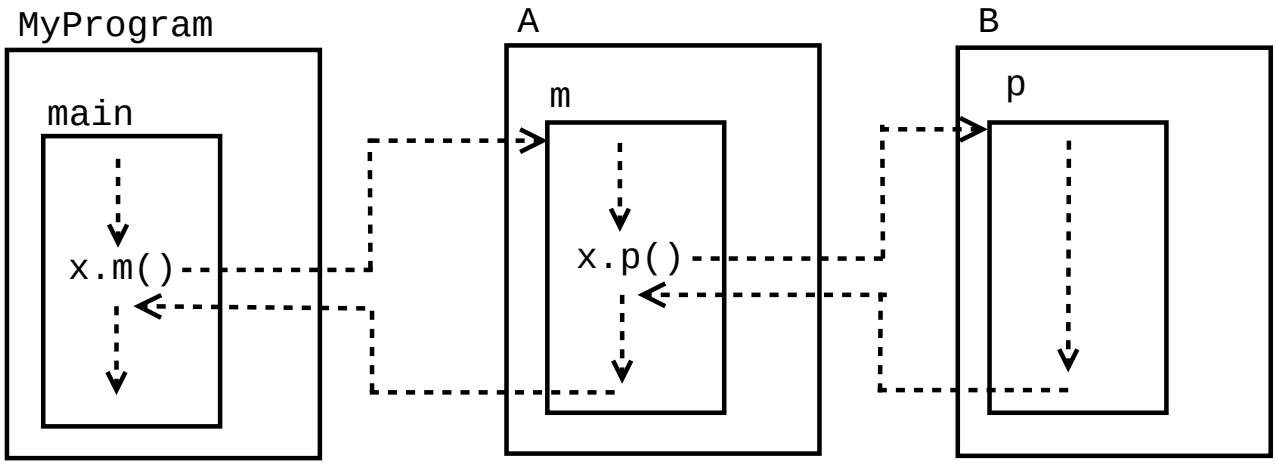
# Program Structure

```
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();
    System.out.println(“Main done”);
  }
}
public class A {
  void m()
  {
    B x = new B();
    x.p();
    System.out.println("m done");
  }
}
public class B {
  void p()
  {
    System.out.println(“Do something”);
  }
}
```

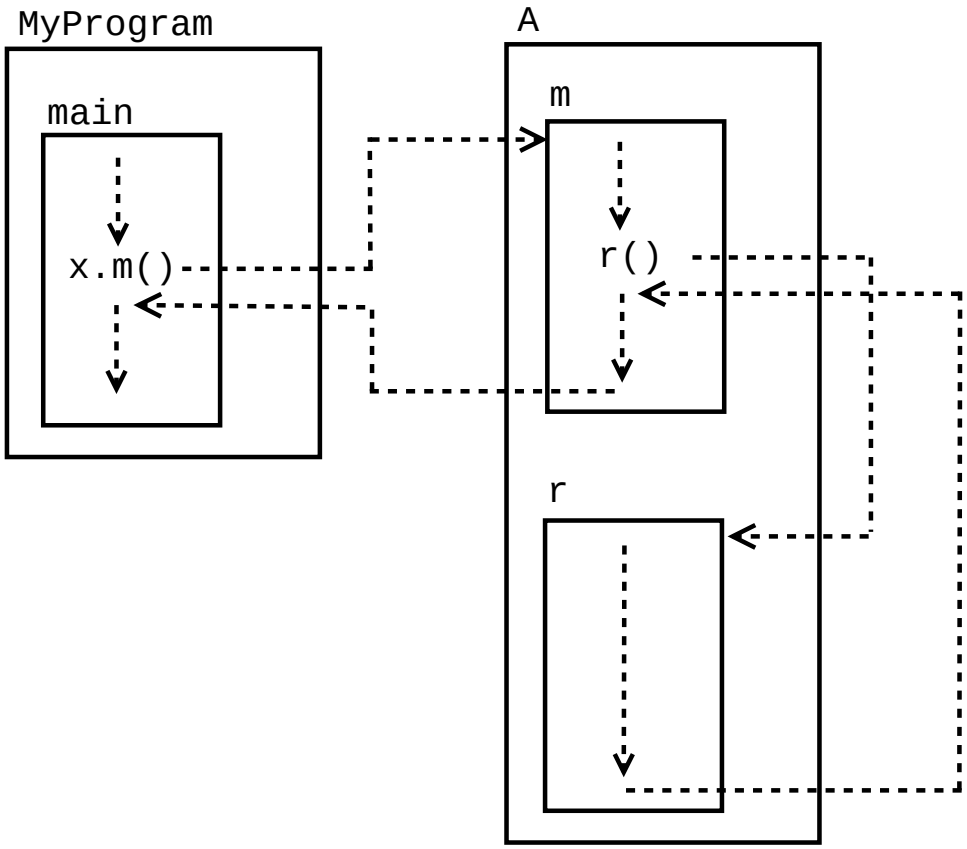
# Method invocation: control flow



# Method invocation: control flow; “this”

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

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

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

# Method invocation: parameter passing

**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*; `this` = *objectreference*;

**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.

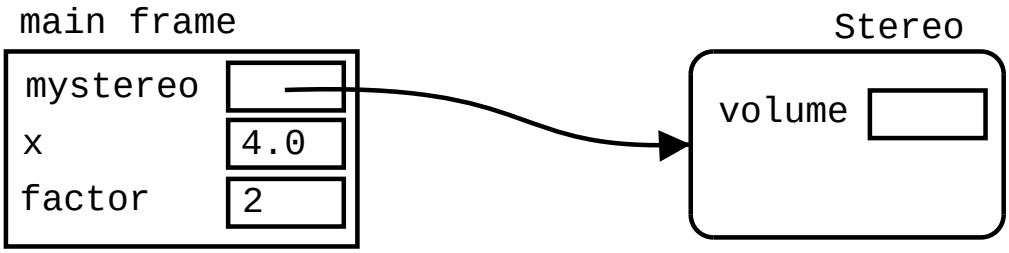
**Fifth:** when the end of the method is reached, or a `return` statement is reached, stop the method, the frame is discarded, and return to the calling method. The calling method is then resumed in the instruction immediatly after the method call.

# 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());
    }
}
```

# Method invocation: Memory structure

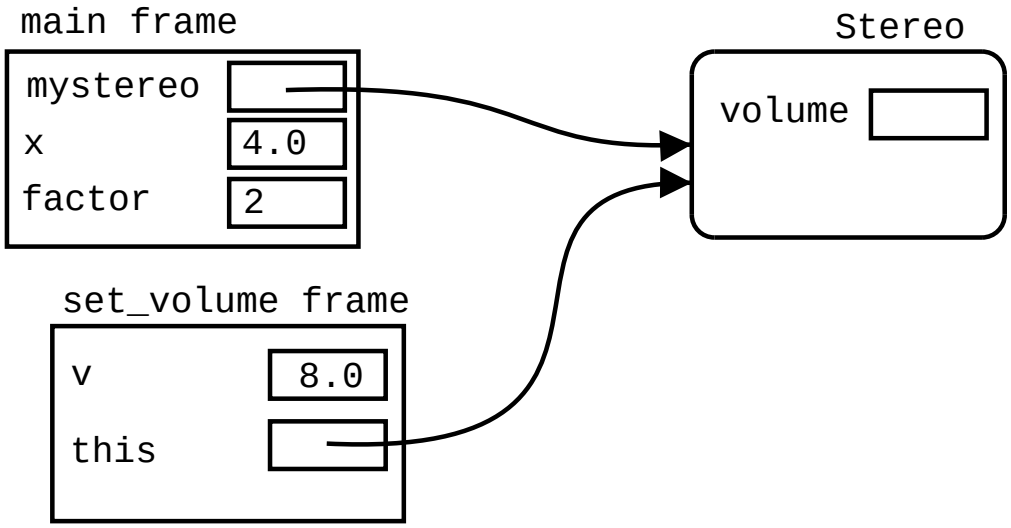
Before calling `mystereo.set_volume(x*factor)`




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

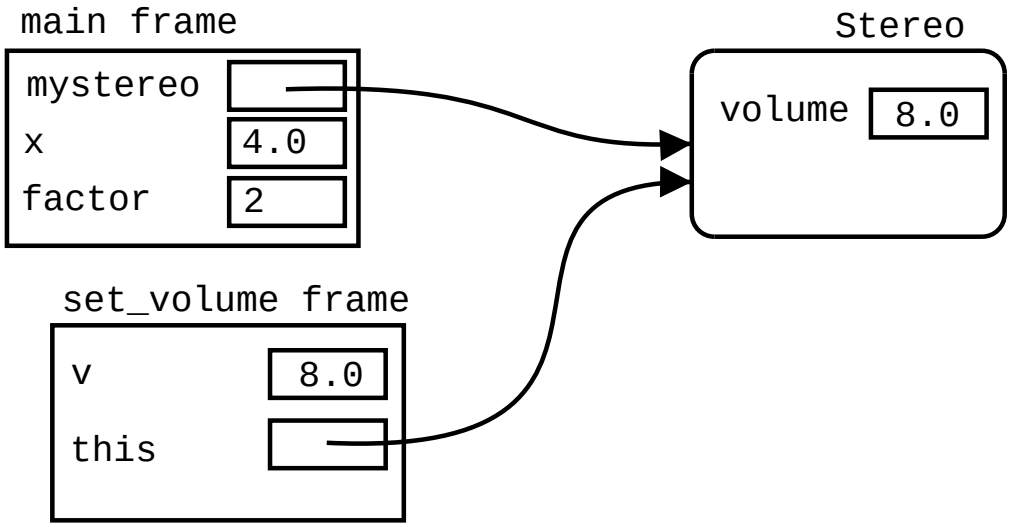
# Method invocation: Memory structure

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



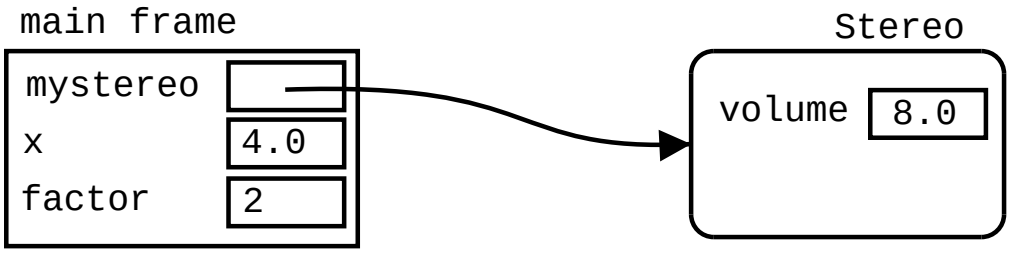
# Method invocation: Memory structure

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):



# Method invocation: Memory structure

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



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

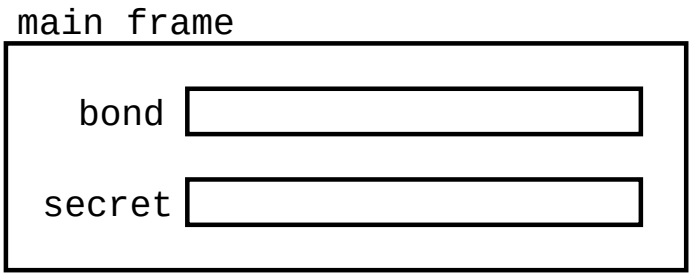
# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

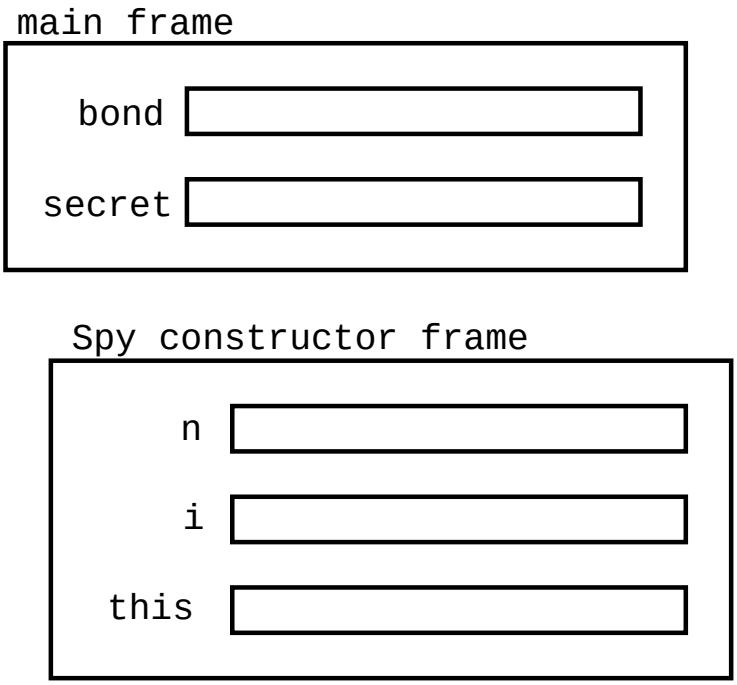
# Method invocation



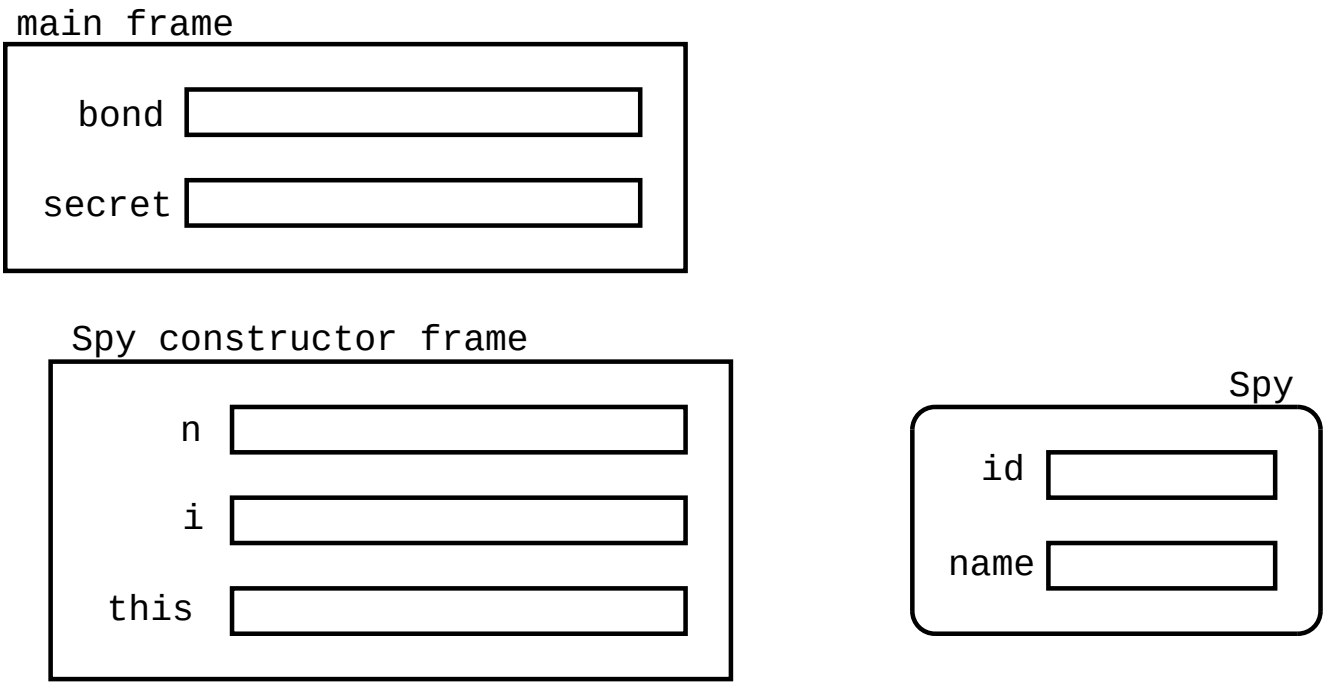
# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

# Method invocation



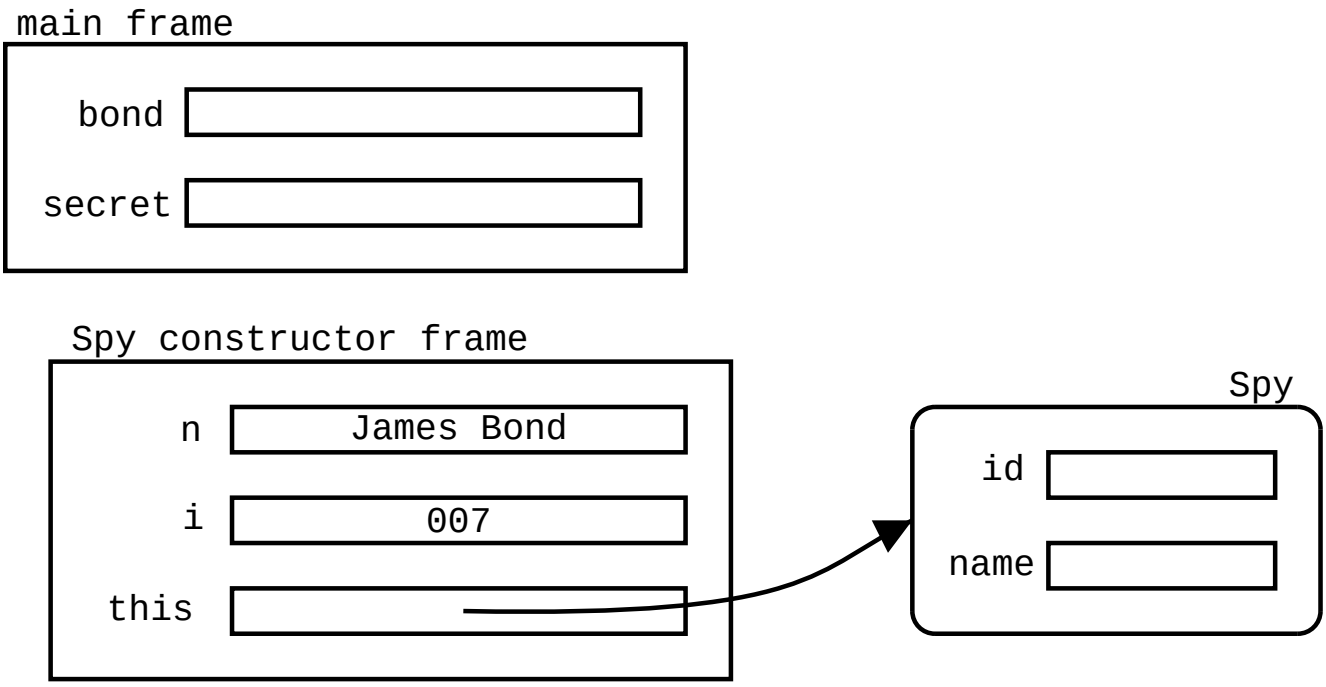
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

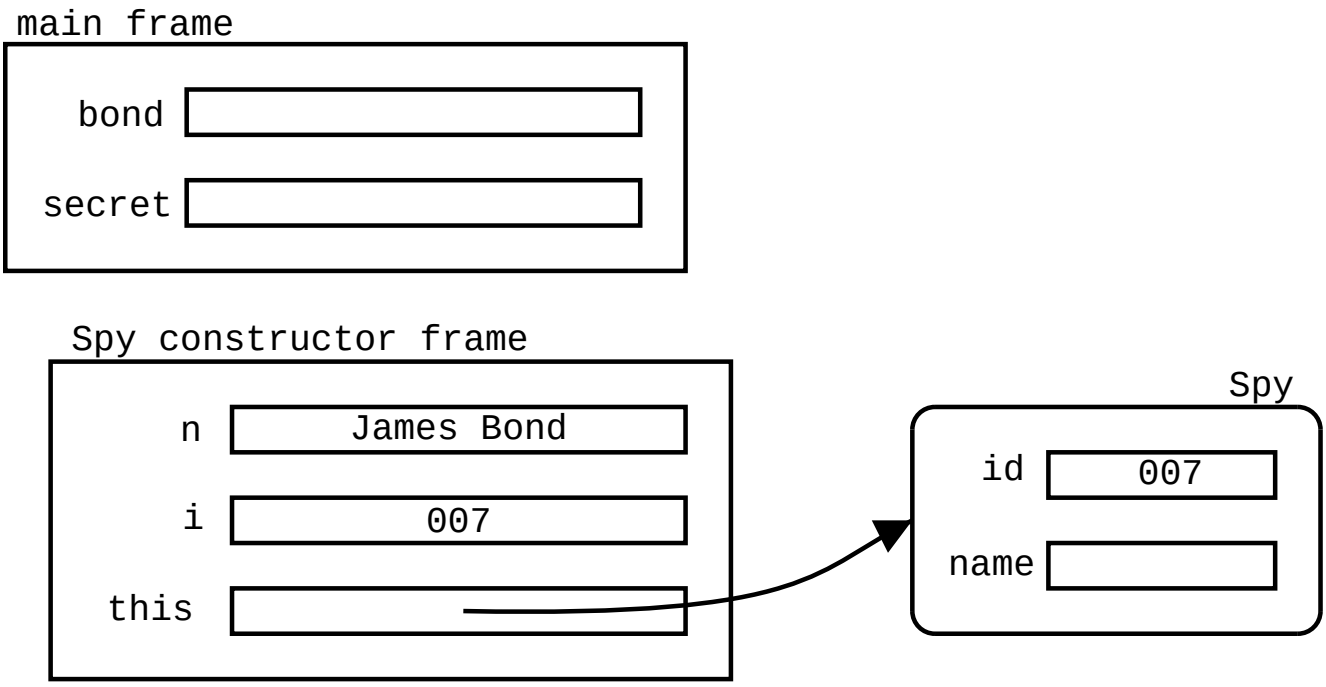
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

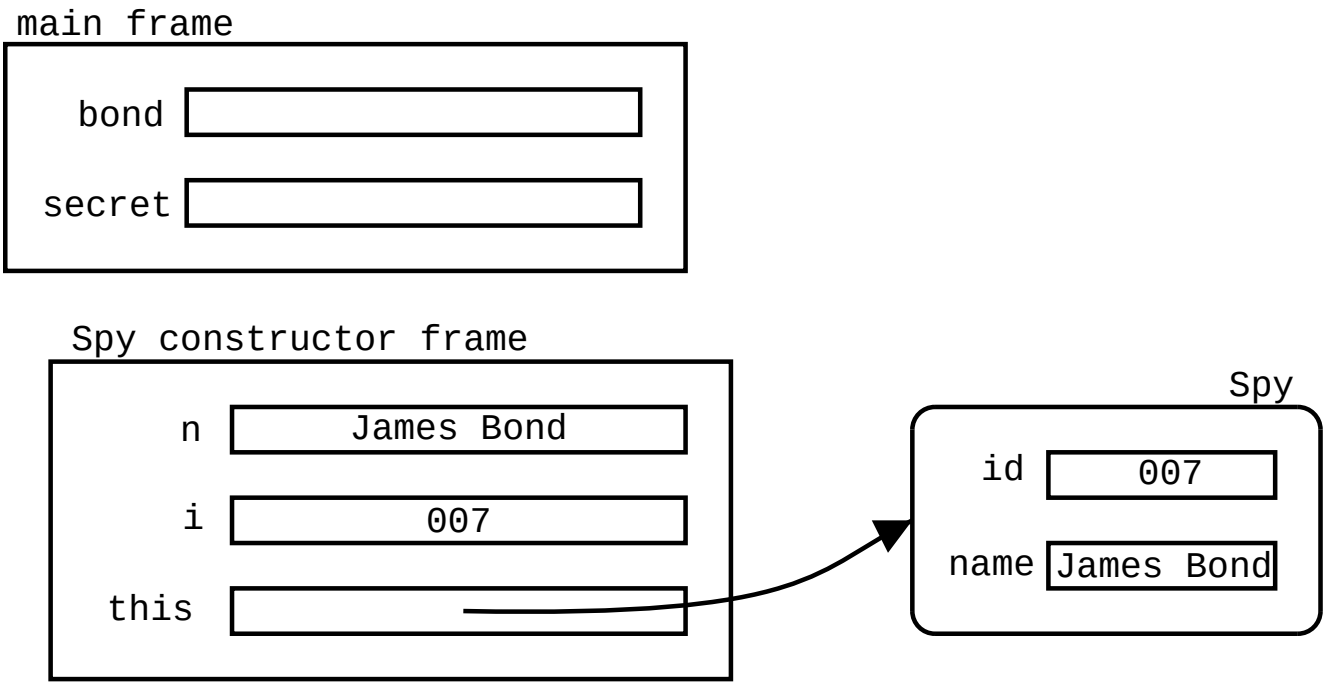
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

# Method invocation



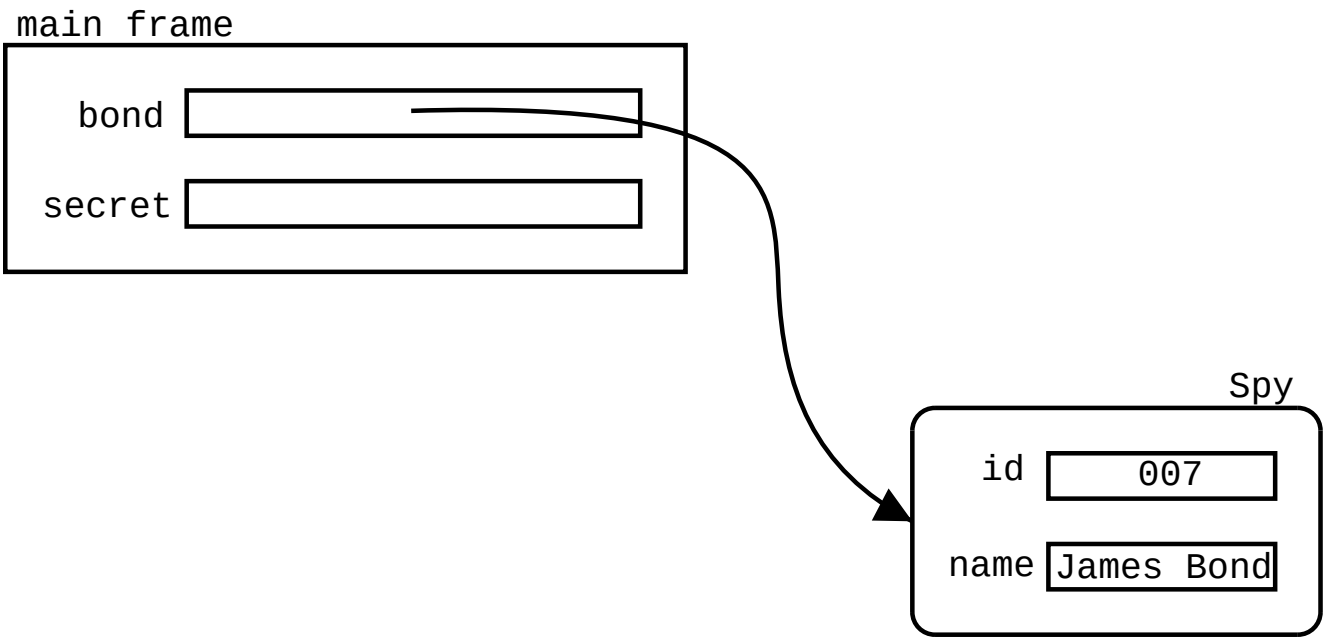
# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

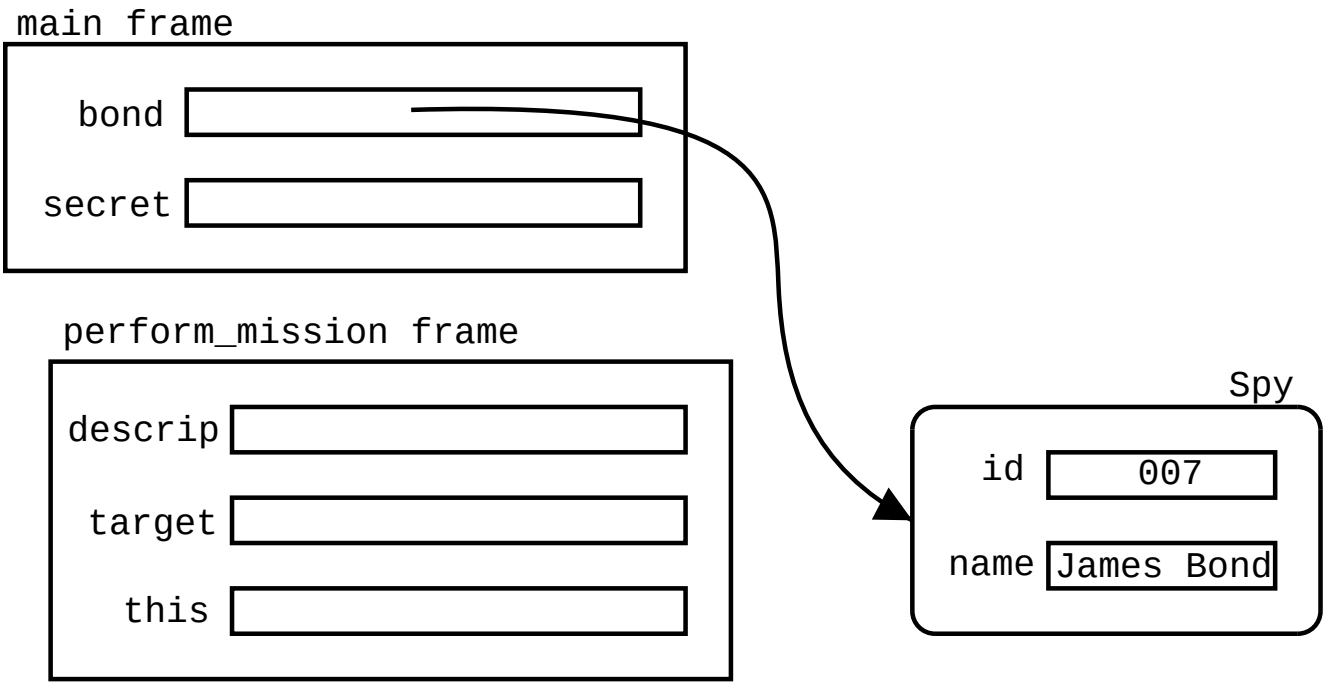
# Method invocation



# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

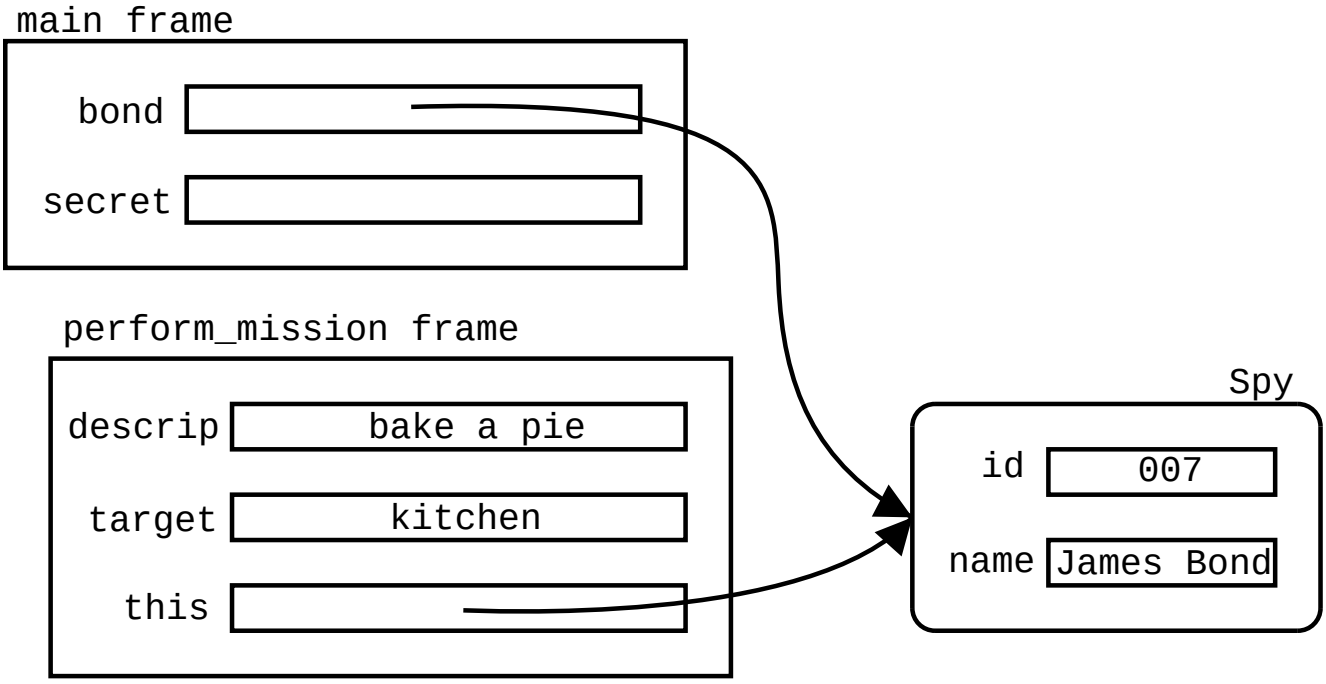
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

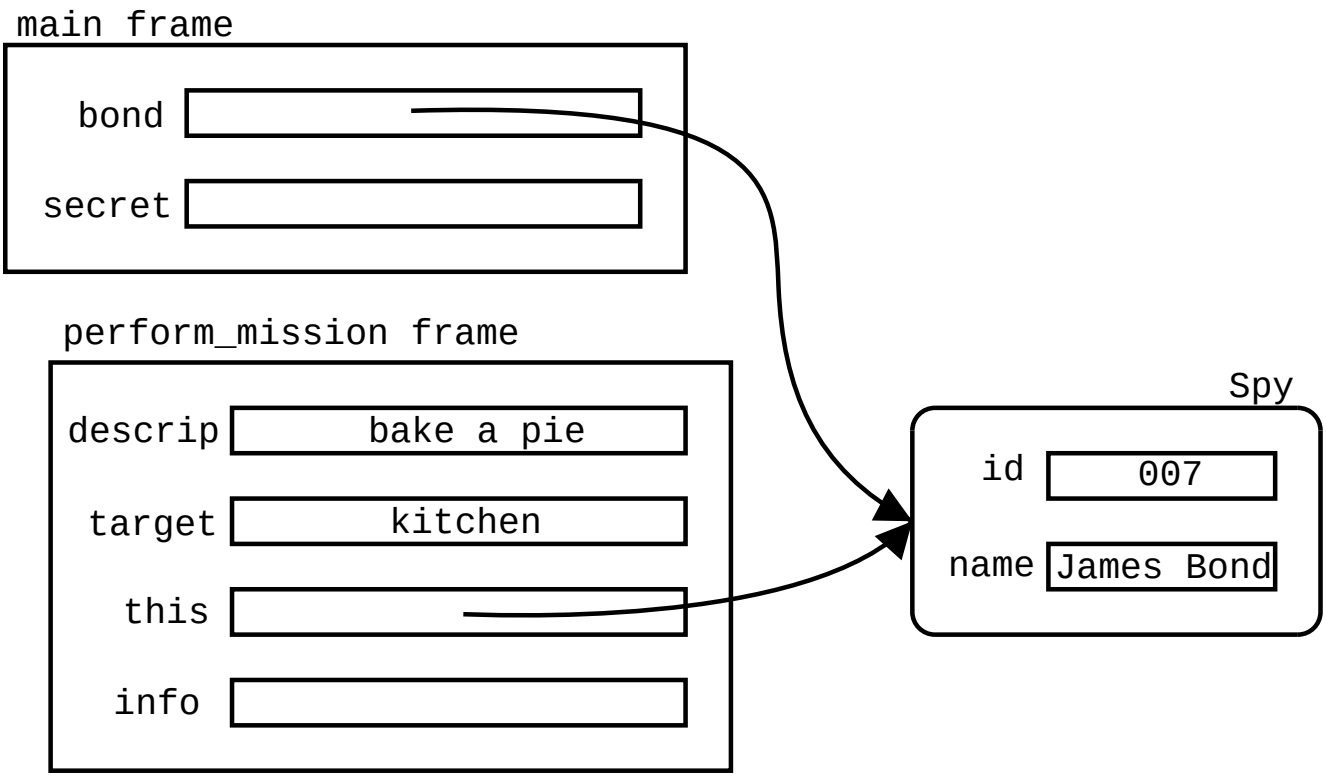
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

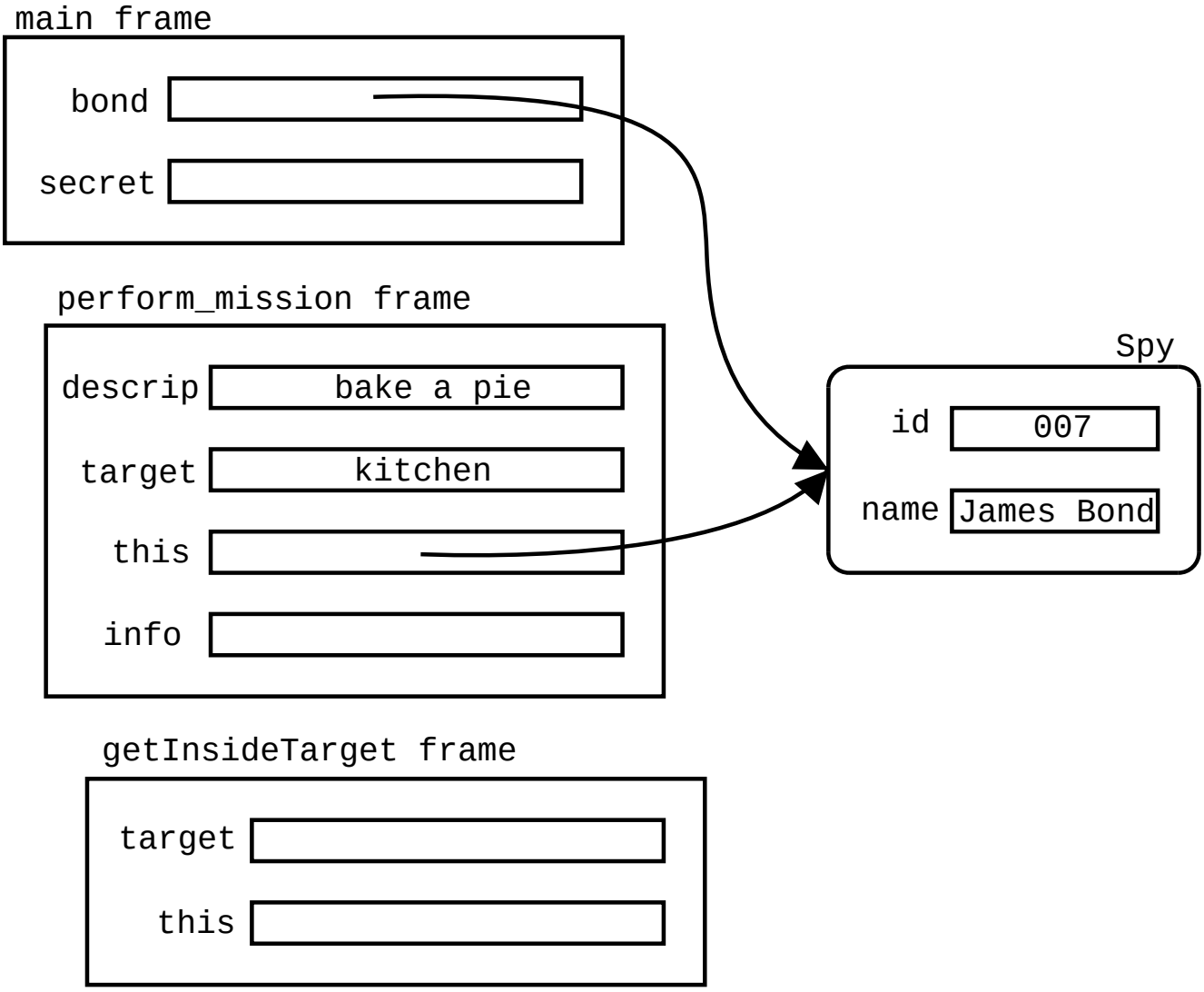
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

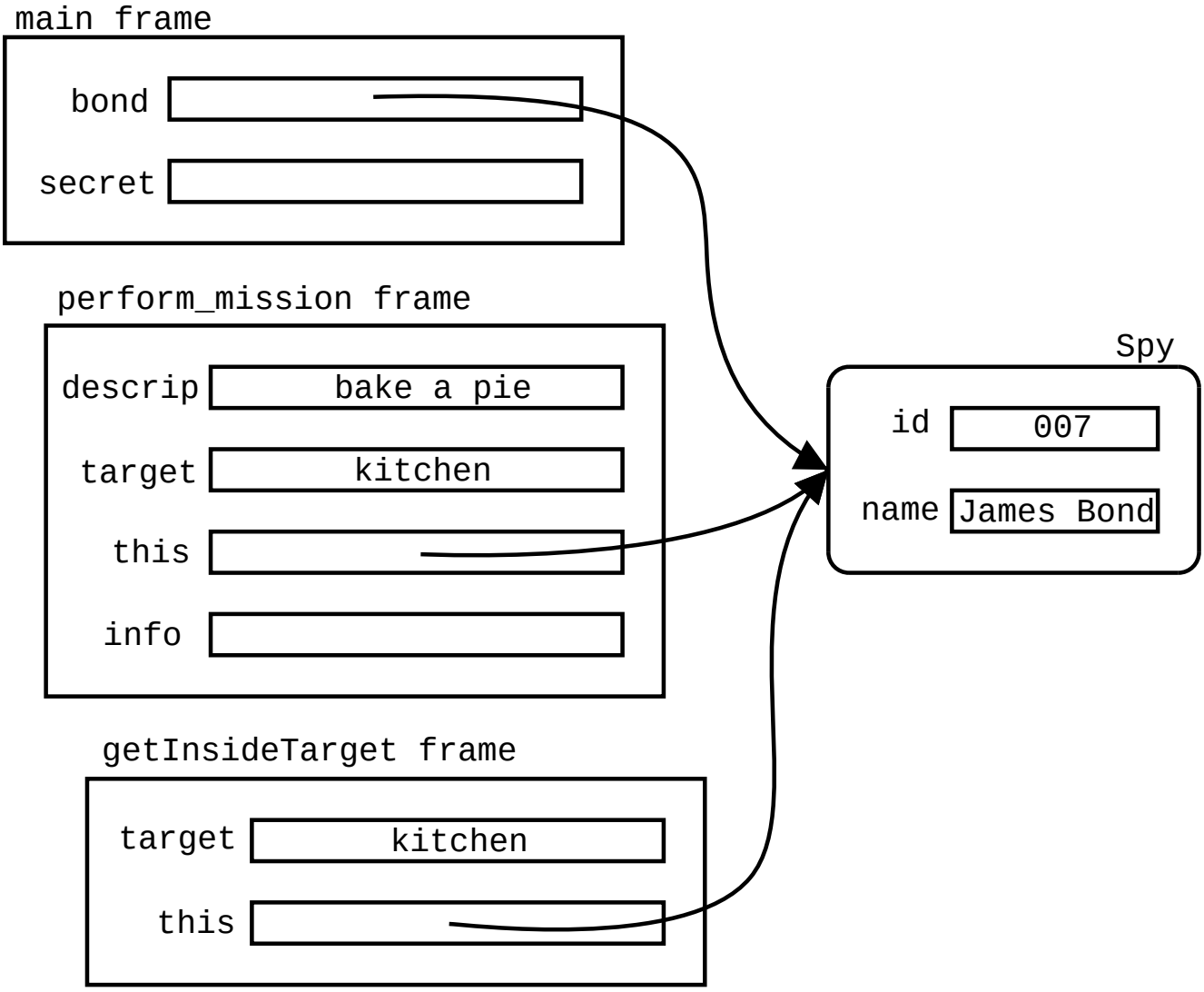
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

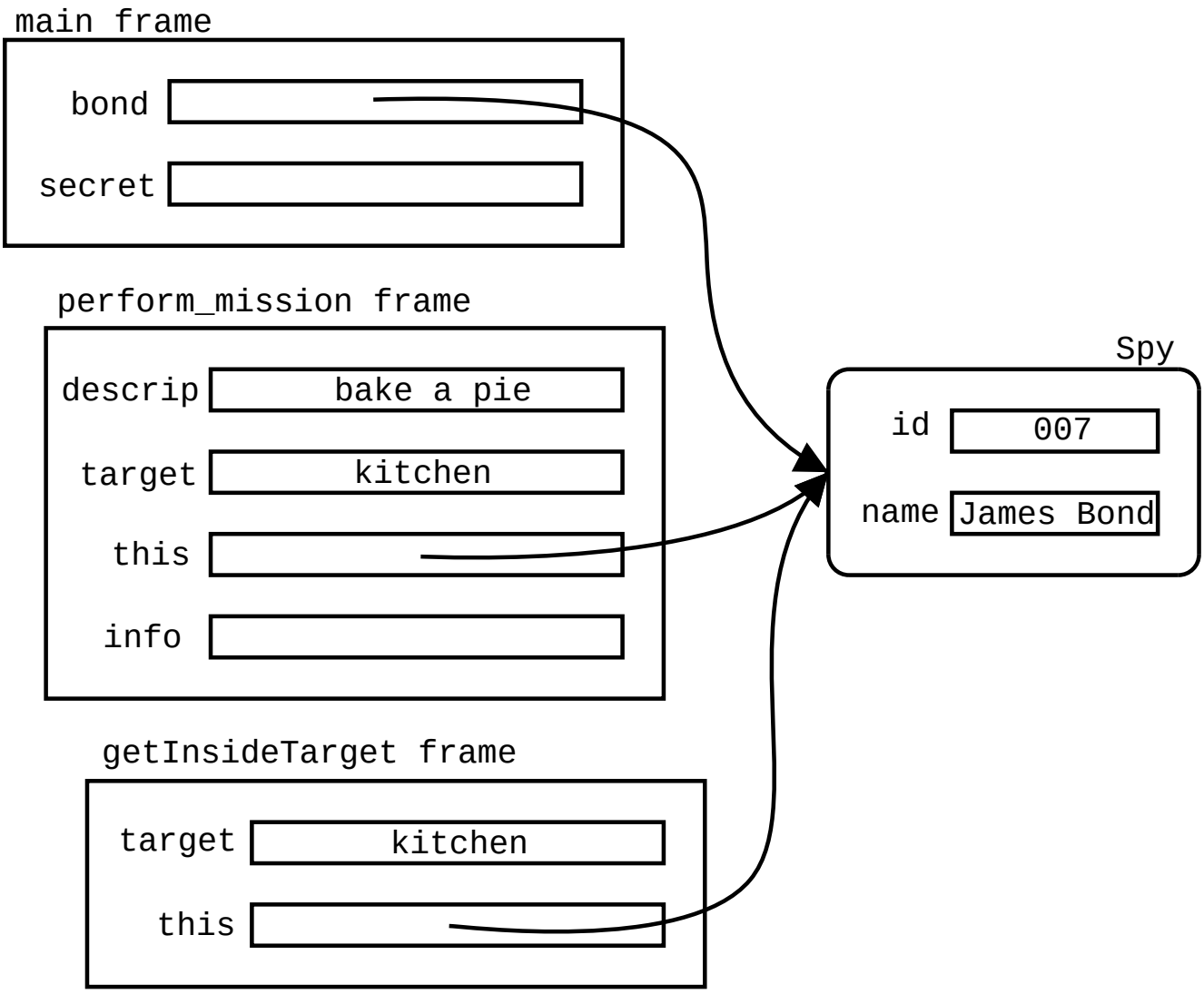
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

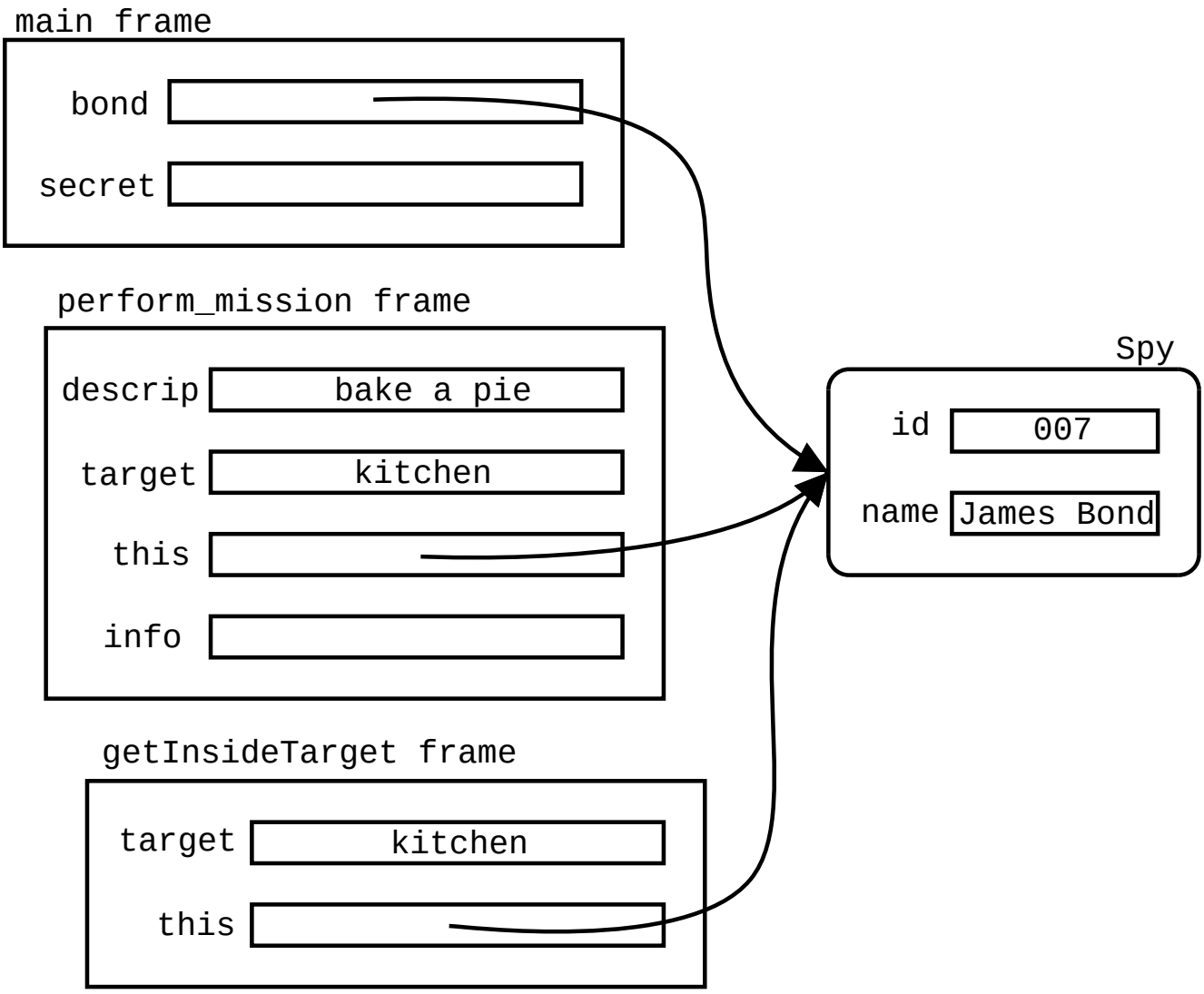
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

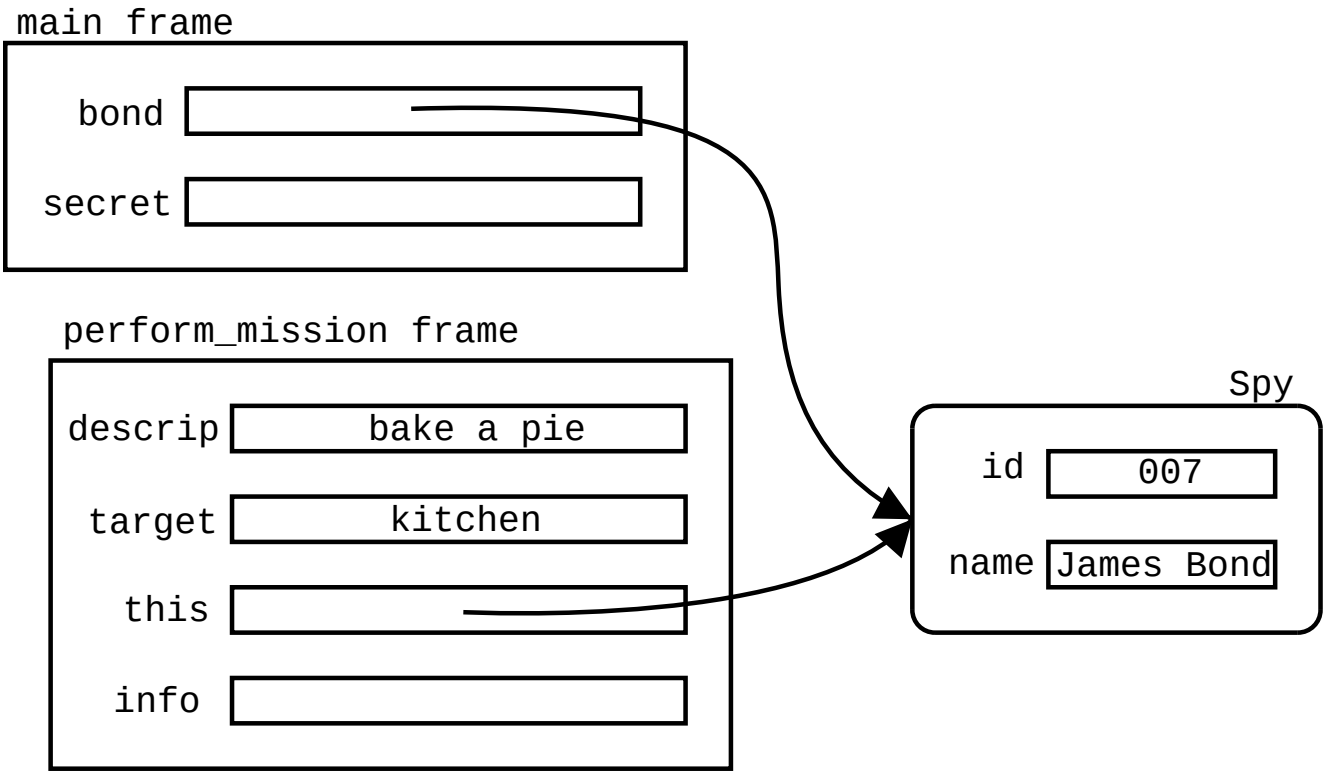
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

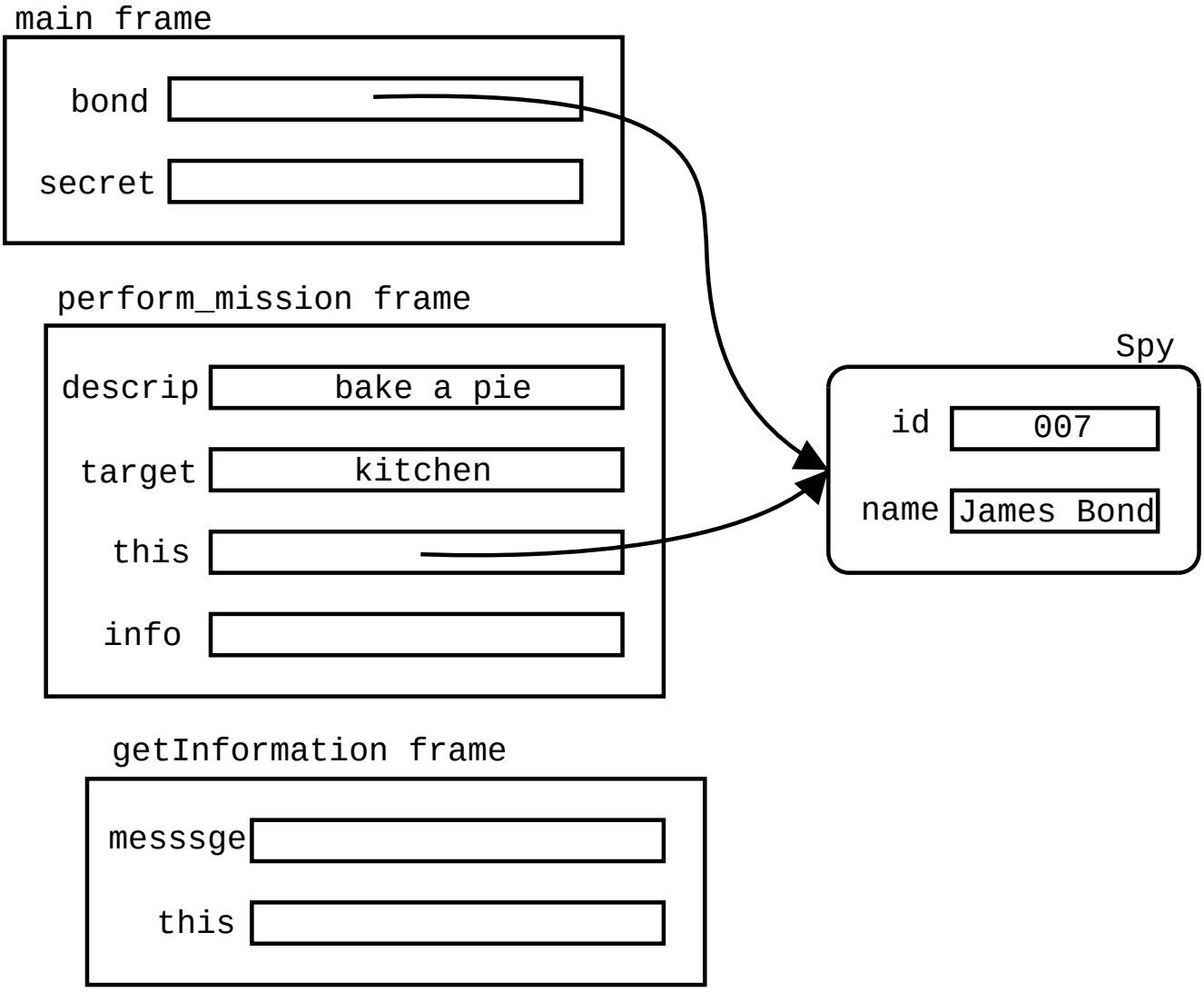
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

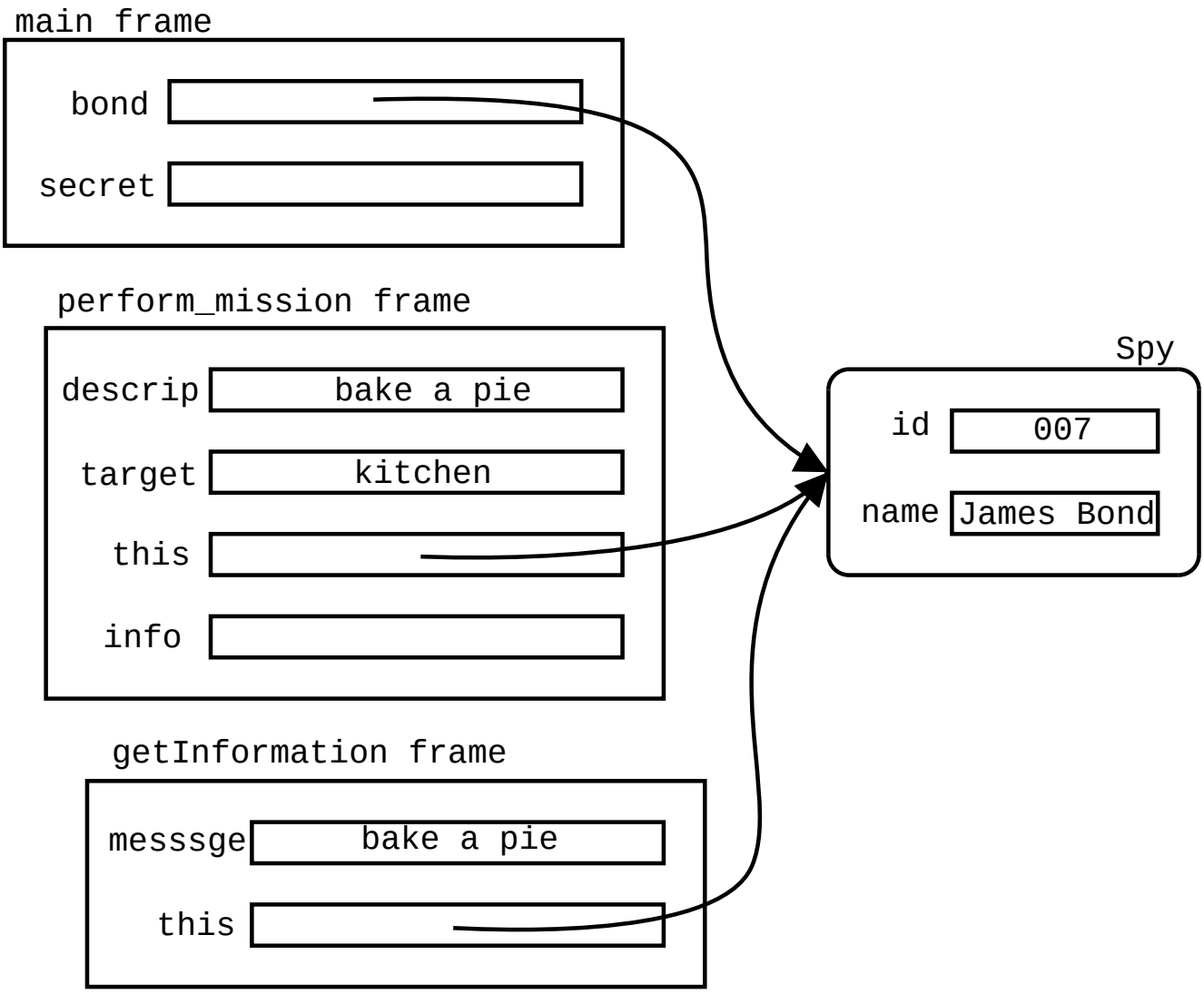
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

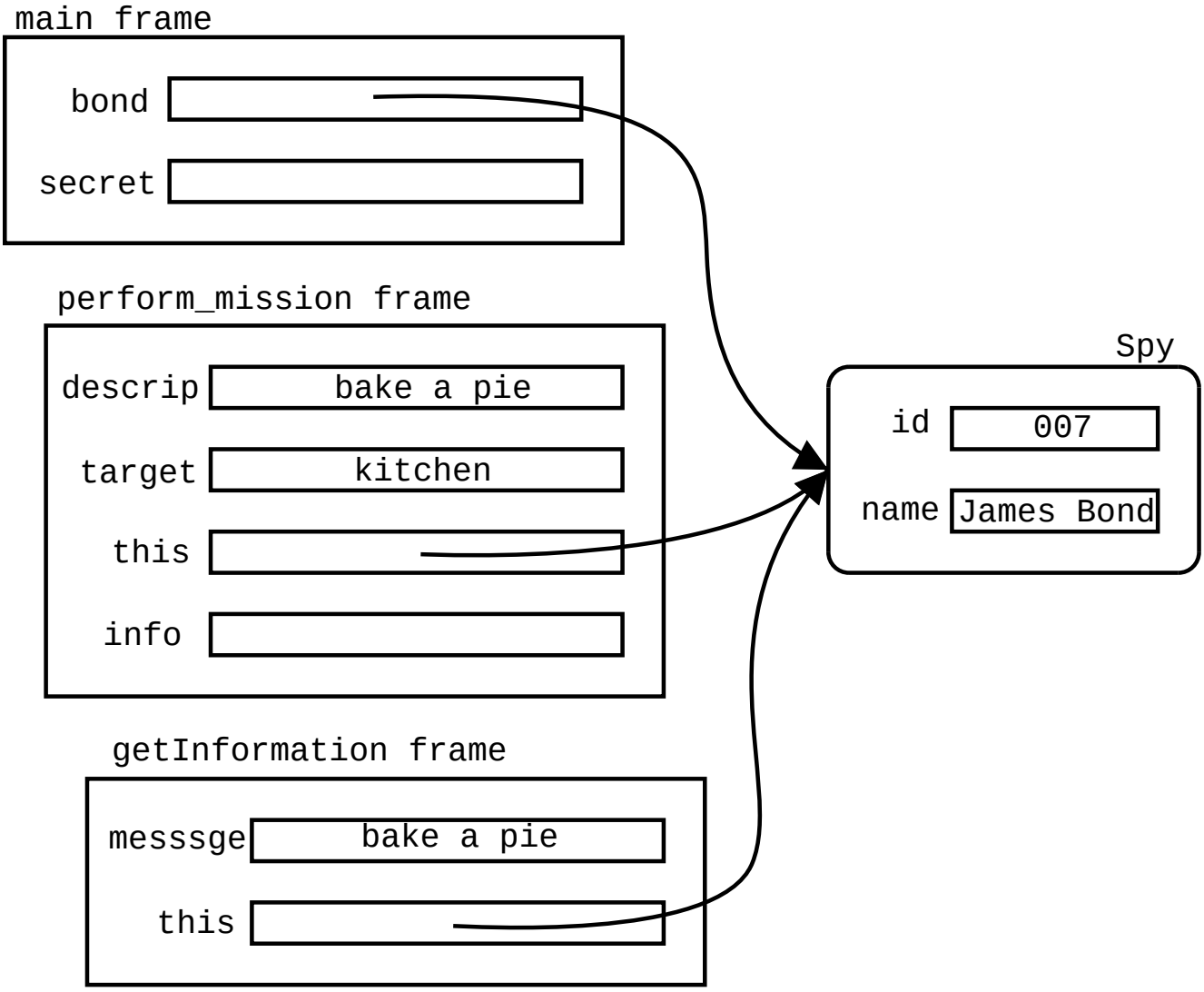
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

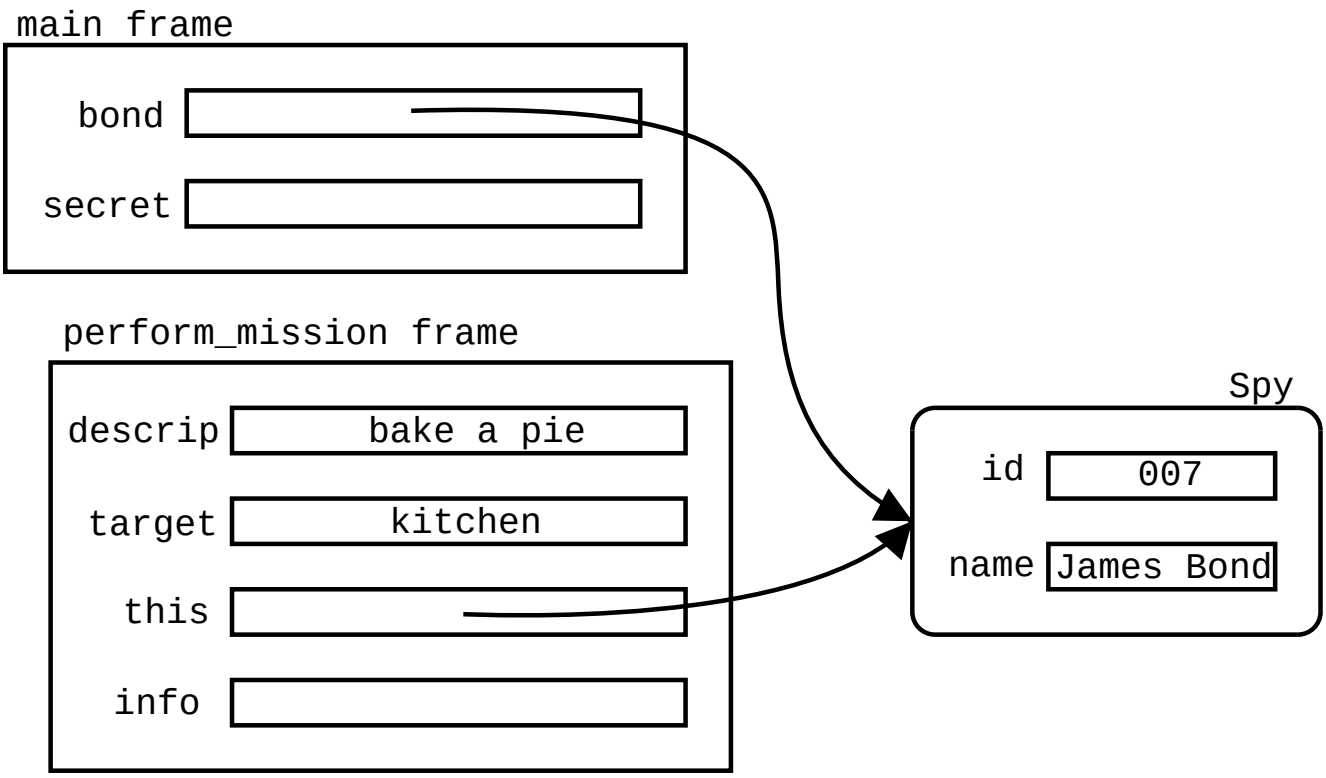
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

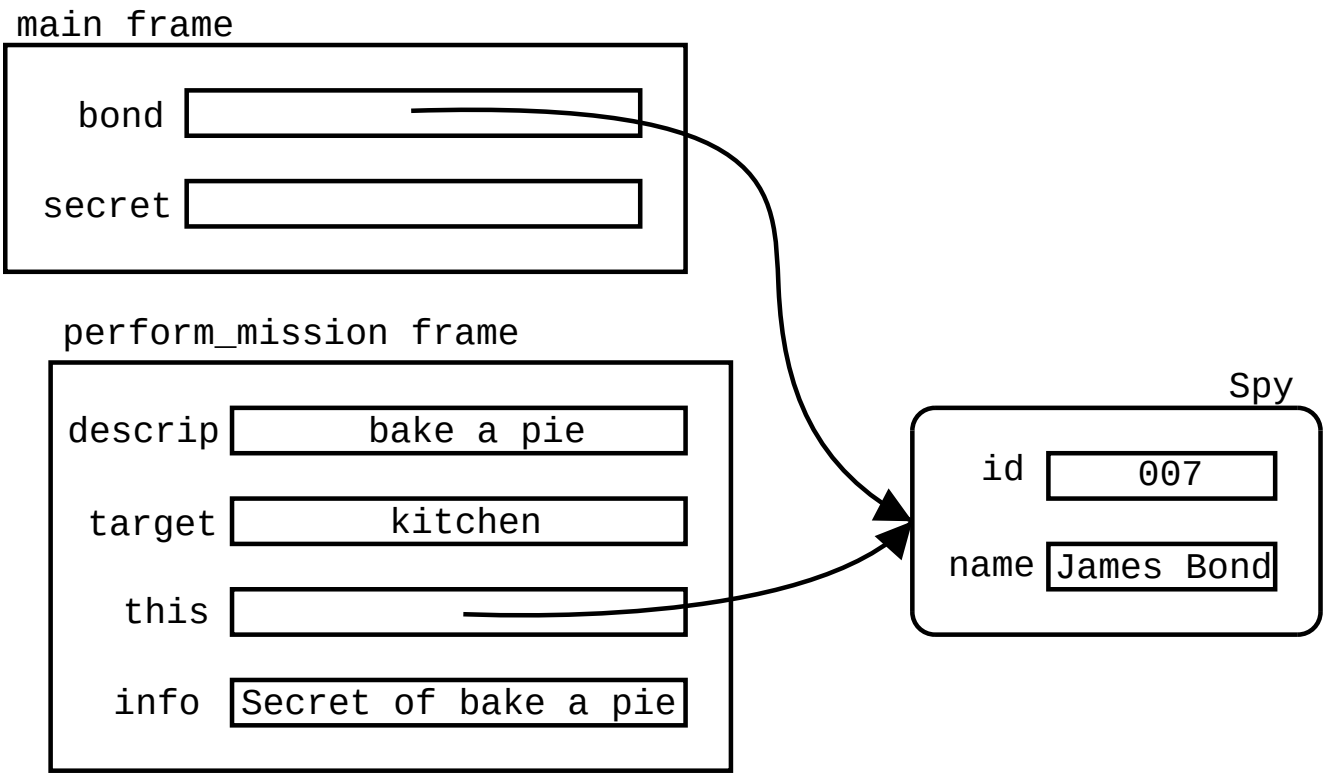
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

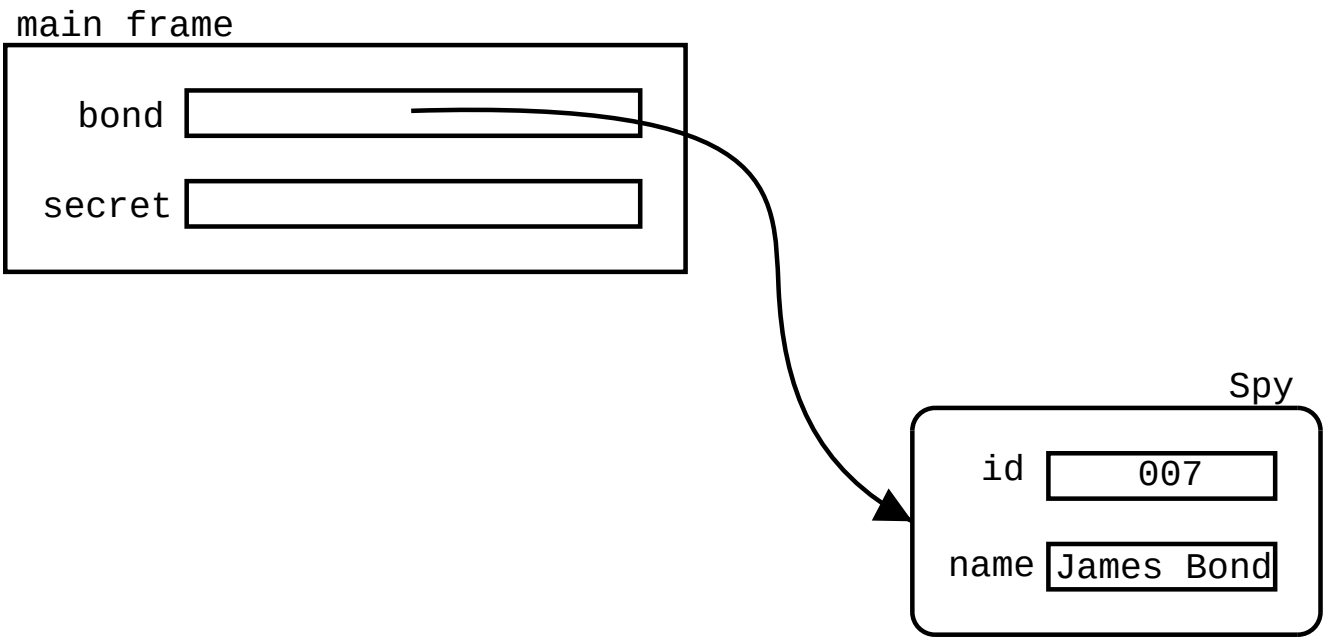
# Method invocation



# Method invocation

```
public class Spy {
  int id;
  String name;
  Spy(String n, int i) {
    id = i;
    name = n;
  }
  String perform_mission(String description,
                         String target) {
    String info;
    this.getInsideTarget(target);
    info = this.getInformation(description);
    return info;
  }
  void getInsideTarget(String target) {
    System.out.println(id + " reporting.");
    System.out.println("Inside: " + target);
  }
  String getInformation(String messge) {
    return "Secret of " + message;
  }
}
```

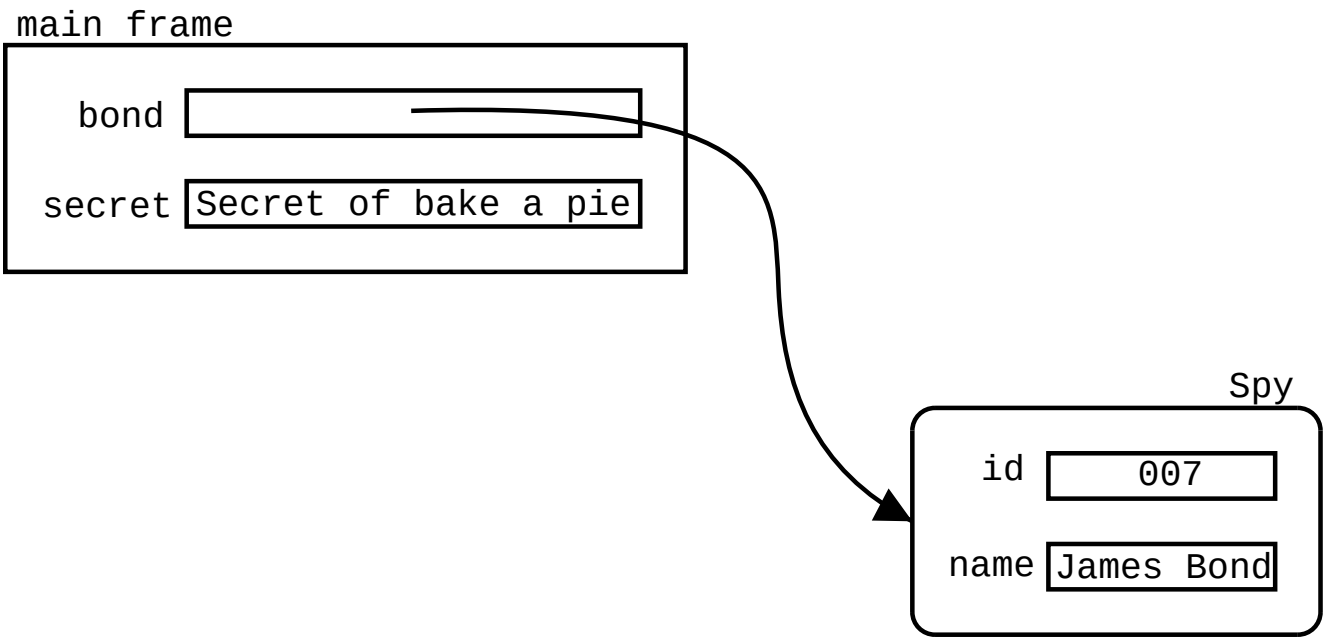
# Method invocation



# Method invocation

```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

# Method invocation



# Method invocation

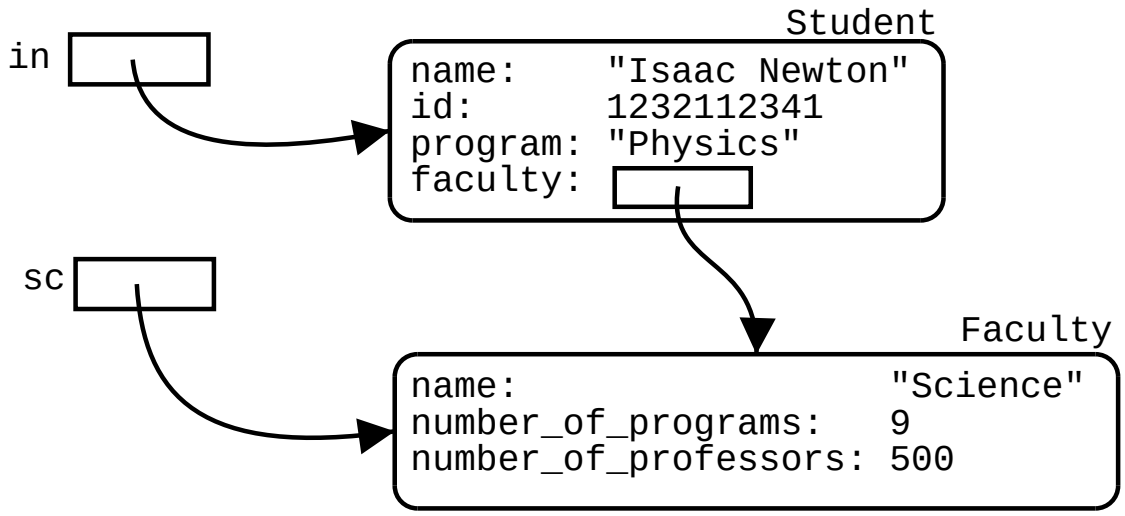
```
public class MI6Sim {
  public static void main(String[] args)
  {
    Spy bond;
    String secret;

    bond = new Spy(“James Bond”, 007);

    secret = bond.perform_mission(“bake a pie”,
                                  “kitchen”);
  }
}
```

# Classes are data types

```
public class Faculty
{
    String name;
    int number_of_programs, number_of_professors;
    //...
}

public class Student
{
    String name;
    long id;
    String program;
    Faculty faculty;
    //...
    void set_prog_and_faculty(String p, Faculty f)
    {
        program = p;
        faculty = f;
    }
    //...
}
```

```
public class StudentDatabase
{
  public static void main(String[] args)
  {
    Faculty sc = new Faculty();

    sc.name = ``Science'';

    Student in = new Student(``Isaac Newton'',
                             1232112341);

    in.set_prog_and_faculty(``Physics'', sc);

    //...
    System.out.println(sc.name);
    System.out.println(in.name);
  }
}
```

# Object structure in memory




```
in.set_prog_and_faculty("Physics", new Faculty());
```

doesn't create the variable `sc`, but then, the `Faculty` object cannot be shared between different `Student` objects.

# 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.)

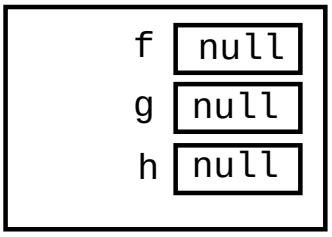
```
public class C {
  public static void main(String[] args)
  {
    A f,g;  // f and g are initialized to null
    B h;    // h is initialized to null
    f = new A();
    // Here f.k is 1
    f.k = 5;
    h = new B();
    h.m();    // assigns a new A to h.x, so ...
    // Here h.x.k is 1
    h.p(f);   // 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.)

```
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

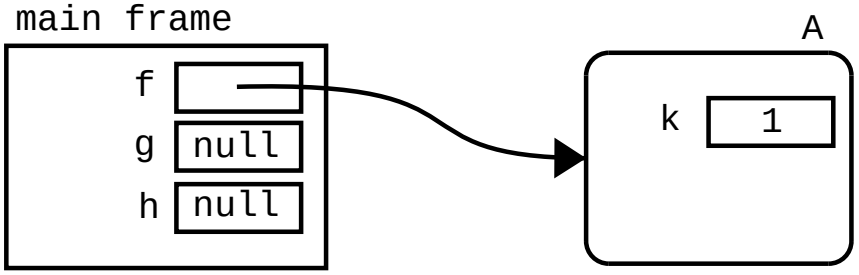


# Example (contd.)

```
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.)

f is assigned a new A object

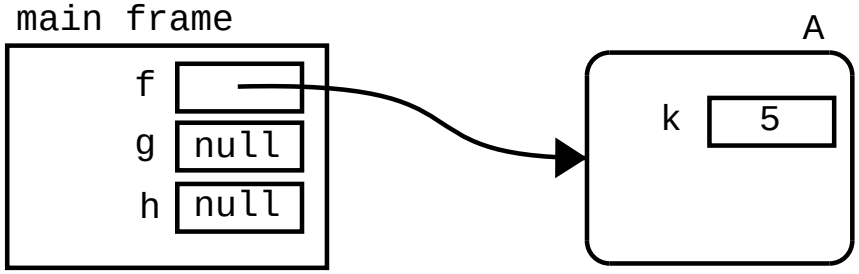


# Example (contd.)

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

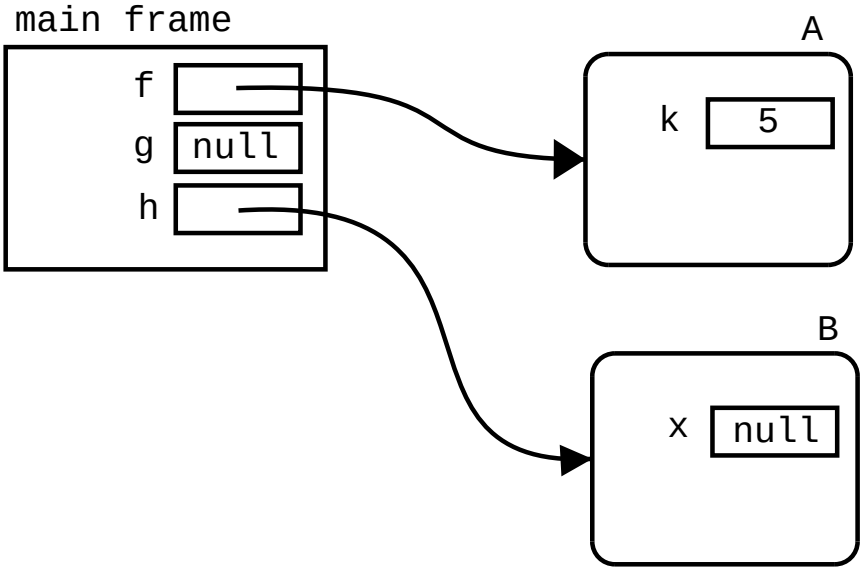


# Example (contd.)

```
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.)

h is assigned a new B

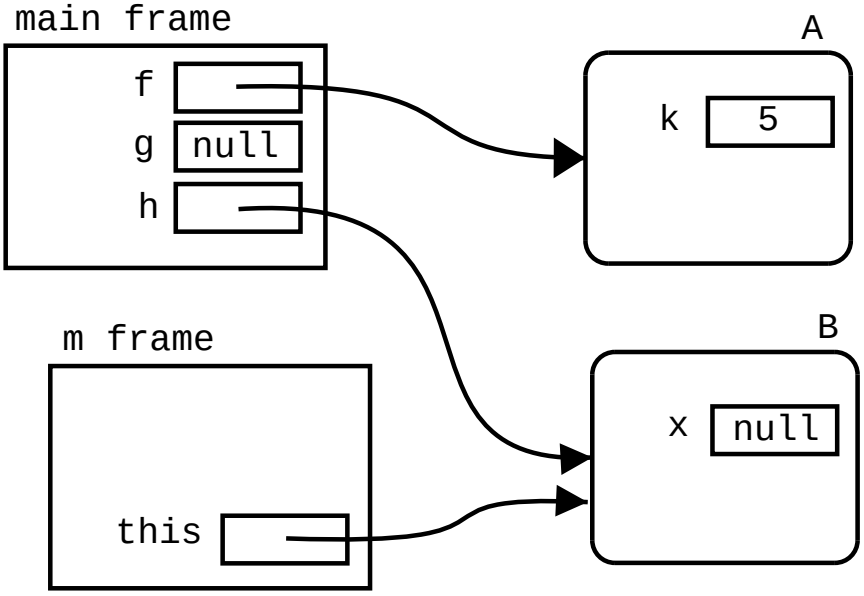


# Example (contd.)

```
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.)

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



# Example (contd.)

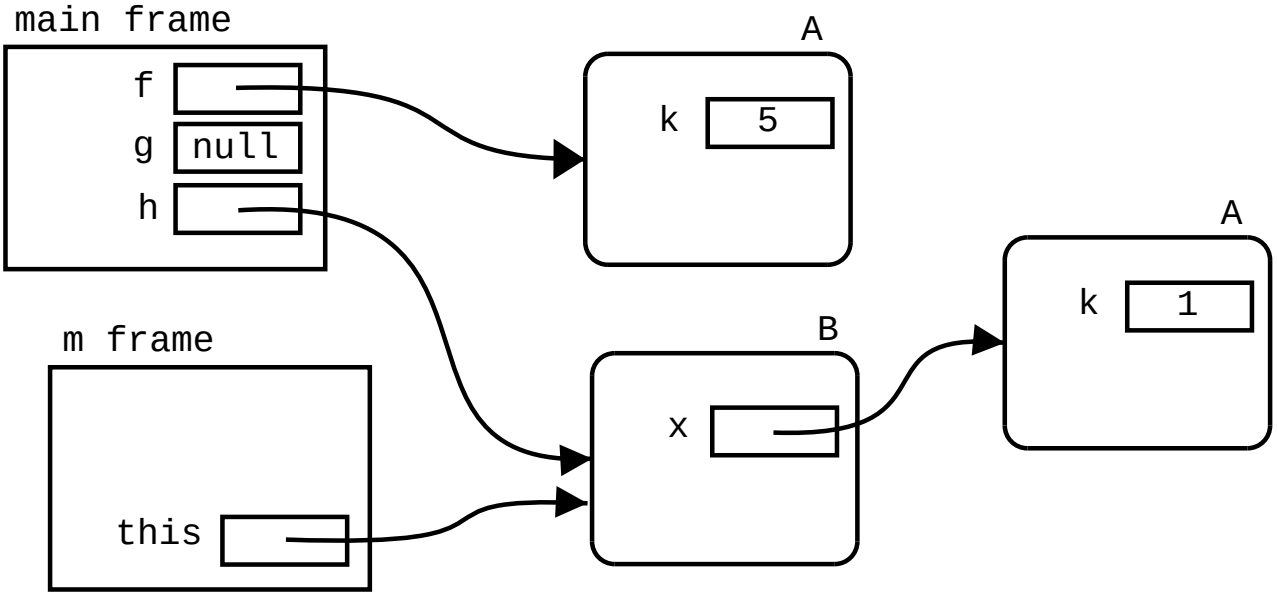
```
public class B {
    A x;          // Objects can be attributes;
    void m()
    {
        x = new A();
    }
    void p(A u) // Parameters may have a class
    {           //    for type
        x = u;  // The object u is created
    }           //    elsewhere
    A r()
    {
        return x;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

# Example (contd.)

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

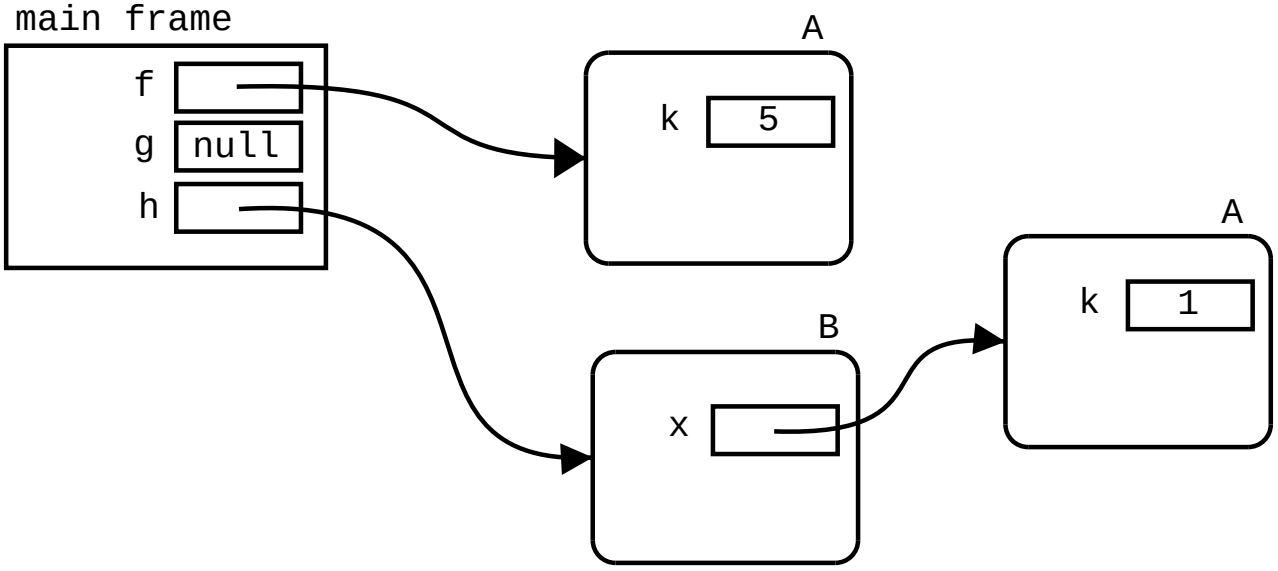
```
x = new A();
```

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



# Example (contd.)

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

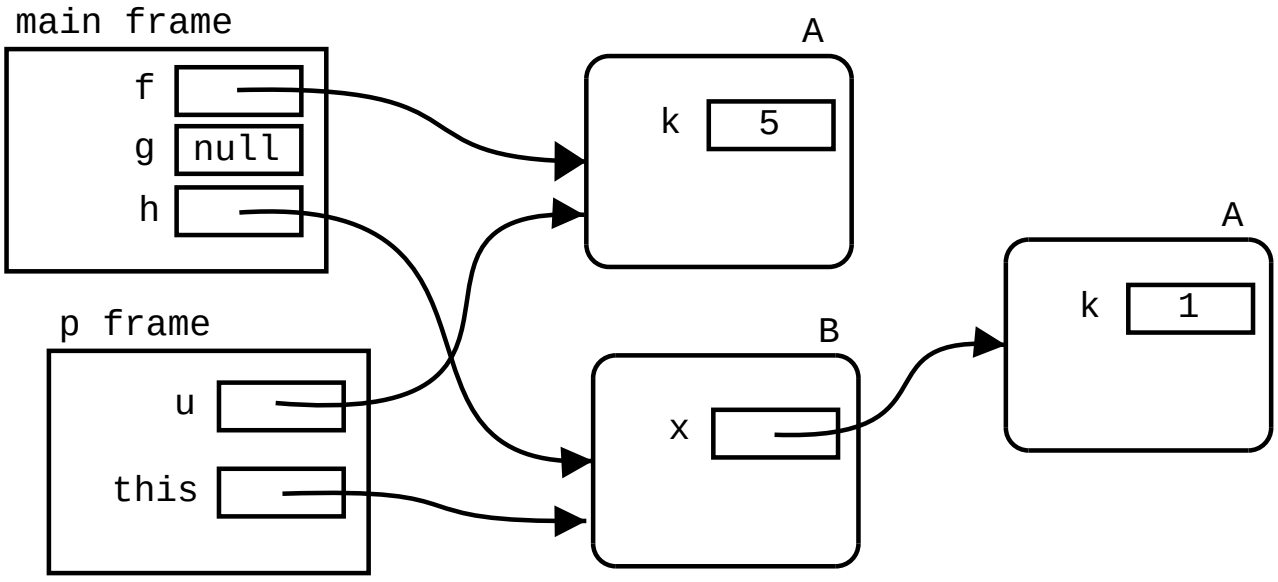


# Example (contd.)

```
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.)

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

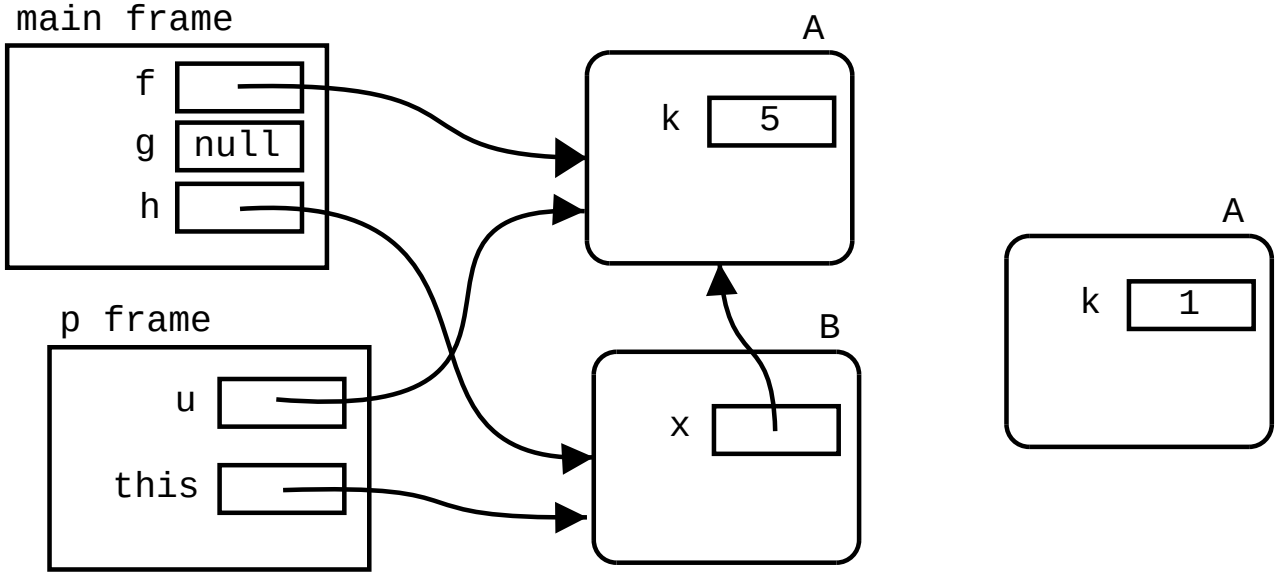


# Example (contd.)

```
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;
    }
}
```

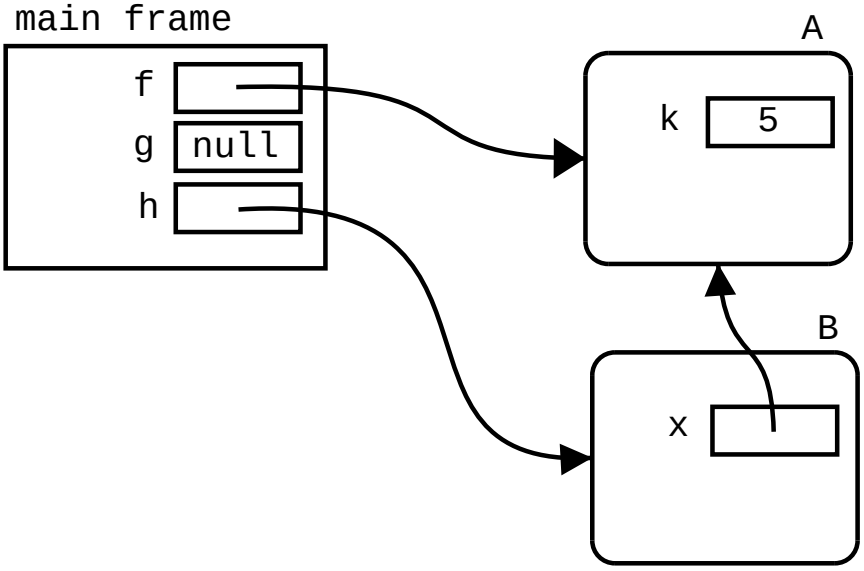
# Example (contd.)

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.

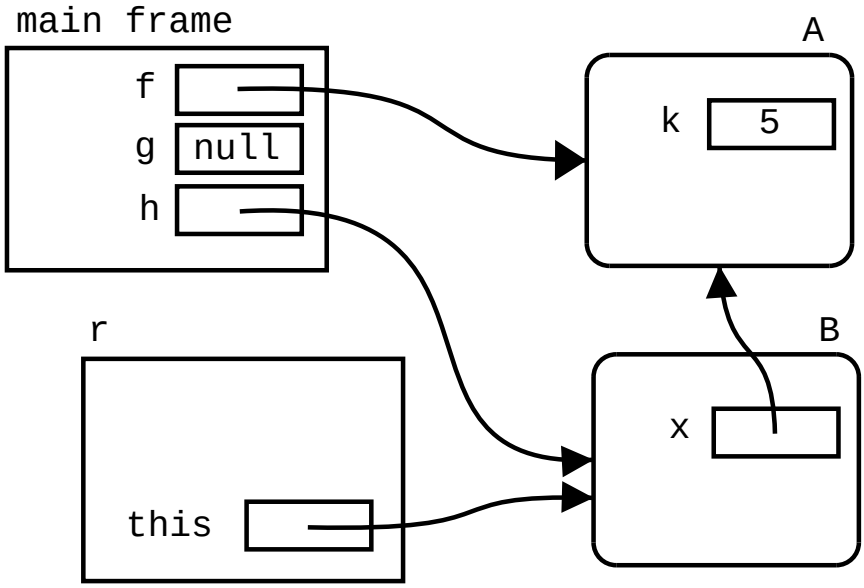


# Example (contd.)

```
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.)

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.

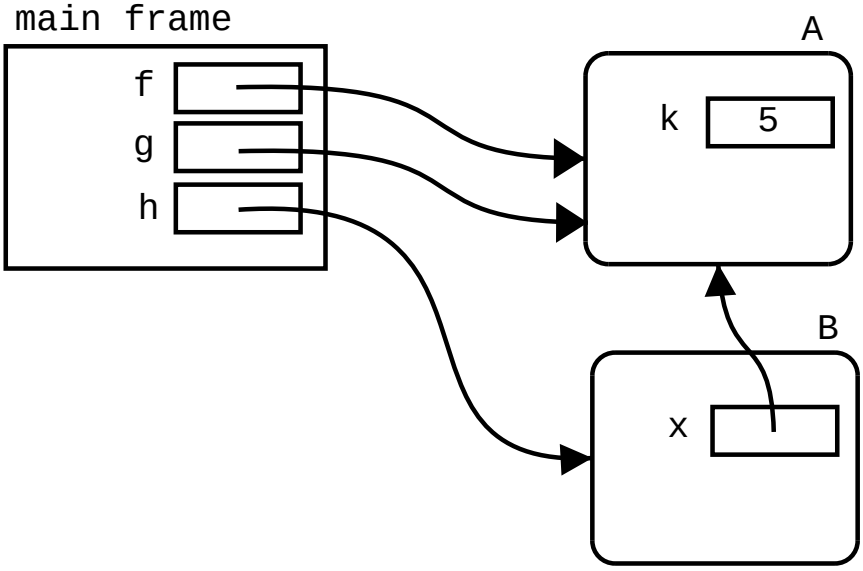


# Example (contd.)

```
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.)

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;`



# Scope

- Different classes can have attributes and methods which have the same names.

- For example given the following class definition

```
public class C {
    int a;
    //...
}
```

the variables `x.a` and `y.a` are different memory locations in the following client:

```
public class D {
    void m()
    {
        C x = new C();
        C y = new C();
        x.a = 3;
        y.a = 5;
    }
}
```

# Scope (contd.)

- This also applies if the attributes are in different classes:

```
public class C {
  int a;
  // ...
}
public class E {
  int a;
  // ...
}
public class D {
  void m()
  {
    C x = new C();
    E y = new E();
    x.a = 3;
    y.a = 5;
  }
}
```

# Scope (contd.)

- The *scope* of a variable, a parameter, an attribute or a method is the part of the program that can access that variable, attribute or method.
- The scope of a parameter of a method is the body of the method.
- Variables declared in the body of a method are called *local variables*, which means that their scope is only the body of the method.
- The direct scope of an attribute of a class or a method is the class itself (e.g. the direct scope of `id` is the `Student` class.)
- However, the indirect scope of an attribute of a class or a method is the rest of the program (e.g. the `id` attribute can be accessed by other clients with the expression `var.id`, where var is of type `Student`.)

# Scope (contd.)

- 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.

# The end