Objects

- An *object* is a *composite* and *reactive* piece of data
  - A piece of *data*: an object is data, it can be treated as a *unit*, a single piece of data
  - *Composite*: an object is a *group* of data
  - *Reactive*:
    - ∗ an object can *react* to *messages* sent to it
    - ∗ we can ask an object to perform a task
    - ∗ we can apply operations on an object

Objects

- A robot *has*: (data)
  - coordinates x and y
  - direction (in radians)
- Given a robot we *can*: (operations/behaviour)
  - make it turn a given angle
  - advance a given distance

Objects and Classes

- The type of an object is a *class*
- A class describes:
  - the structure of its objects (attributes)
  - and its operations (methods)
- A class is ***not*** the same as an object
- A class is like the "blueprint" of a family of objects
- An object is a particular *instance* of a class

Classes

- Classes have a dual role in Java:
  - They are the data-type of *objects*
  - They are modules
- A single class alone doesn't do anything ...
  - A class is useful in a context of other classes

Objects and Classes

- Defining classes

```
public class Classname
{
  // Attributes
  // Methods
}
```

- Creating objects of a defined class

```
Classname variable;
variable = new Classname(parameters);
```

- Sending a message to an object

```
variable.method_name(arguments);
```

Objects and Classes

- Defining classes

```
public class Classname
{
  // Attributes
  // Methods
}
```

- Declaring an attribute

```
type identifier;
```

- Declaring a method

```
void method_name (parameters)
{
    // body
}
```

Objects and Classes

- Declaring a method that does not return information

```
void method_name (parameters)
{
    // body
}
```

- Declaring a method that does return information

```
type method_name (parameters)
{
    // body
    return expression;
}
```

Objects and Classes

```
public class Robot
{
    double x, y, direction;

    Robot (double dir)
    {
        x = 0.0;
        y = 0.0;
        direction = dir;
    }

    void turn(double angle)
    {
        direction = direction - angle;
    }

    // Continues below
    void advance(double distance)
    {
        double dx, dy;
        dx = distance * Math.cos(direction);
        dy = distance * Math.sin(direction);
        x = x + dx;
        y = y + dy;
    }
}  // End of Robot class
```

Objects and Classes

```java
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto, marc;
        ernesto = new Robot(Math.PI/2);
        marc = new Robot(0.0);
        ernesto.advance(200.0);
        marc.turn(Math.PI / 2);
        marc.advance(150.0);
    }
}
```

Objects and Classes

- Each object has its own separate *identity*, its own individual *state*

- The *state* of an object is the current value of its attributes

- The state of an object can change:

  - when we ask the object to do something
  - ... therefore, the methods of the object's class are responsible for changes to the object's state

Parameters

- *Parameters*: variables that receive information necessary to execute a method

- Information flow:
  - when a method is invoked,
  - the caller *passes* information to the method in the form of *arguments*
  - and the method receives that information in its parameters

Objects and Classes

```
import java.util.Scanner;
public class Test
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        System.out.print(‘‘Enter a distance: ‘‘);
        double x = scanner.nextDouble();
        ernesto.advance(x);
    }
}
```

Objects and Classes

```java
import java.util.Scanner;
public class Robot
{
    double x, y, direction;

    Robot (double dir) { ... }

    void turn(double angle) { ... }

    void advance(double distance)
    {
        double dx, dy;
        dx = distance * Math.cos(direction);
        dy = distance * Math.sin(direction);
        x = x + dx;
        y = y + dy;
    }
}
```

Objects and Classes

```
import java.util.Scanner;
public class Robot
{
    double x, y, direction;

    Robot (double dir) { ... }

    void turn(double angle) { ... }

    void advance()
    {
        double distance;
        Scanner scanner = new Scanner(System.in);
        System.out.print(“Enter a distance: ”);
        distance = scanner.nextDouble();
        double dx, dy;
        dx = distance * Math.cos(direction);
        dy = distance * Math.sin(direction);
        x = x + dx;
        y = y + dy;
    }
}
```

Objects and Classes

```
import java.util.Scanner;
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        ernesto.advance();
    }
}
```

Objects and Classes

- The previous two slides are an example of bad design
- Important design question:
  - Where do I put the code that asks the user for information?
- Answer:
  - Separate the parts of the program that interact with the user from the parts that actually do computation
  - Don't replace a parameter by statements asking the user

Methods

- A method is a procedure that performs a task
- A method may have some inputs (its parameters)
- A method may have an output (its returned value)
- Never confuse the parameters with user-input
  - The main reason is that sometimes we want a method to receive its input not from the user but from another part of the program
- Each method on its own is an algorithm (a procedure) that may be a subproblem from a larger problem

Good (do)

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
}
```

Bad (don't)

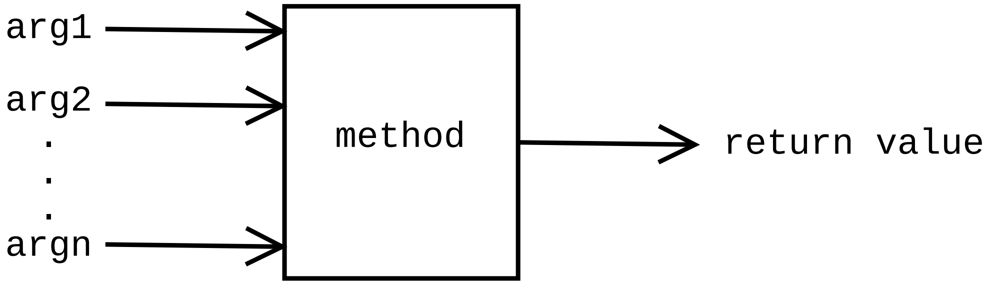
```
void advance()
{
    double distance;
    Scanner scanner = new Scanner(System.in);
    System.out.print(“Enter a distance: ”);
    distance = scanner.nextDouble();
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
}
```

Objects and classes

- Rules on using objects:
  - Before applying methods to an object, the object has to exist (it must be created)
  - If a method is applied to an object, then:
    - the method must be defined in the object's class
    - the number of arguments passed must be the same as the number of parameters expected
    - the types of arguments passed must match the types of the parameters, in the same order

Methods as functions

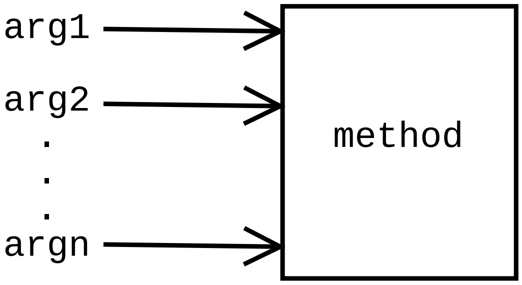
- Methods can be viewed as a "black box" with inputs and outputs:



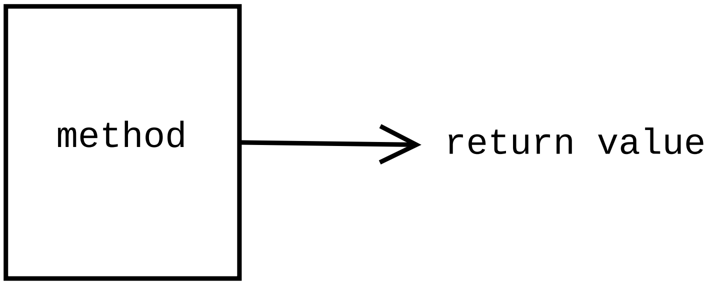
- There are several kinds of methods:
  - Constructors: Initialize a newly created object.
  - Mutators (setters): Modify the state of objects,
  - Accessors (getters): Return information about the object,
  - Others:
    - Modify the state and return information
    - Relay messages to other objects
    - Mixed

Method types

- Mutators are usually `void` methods, which do not return anything, but modify the state of the object:



- Accessor methods may only return values without expecting any arguments as input:



Constructors

- The *constructor* of a class is a special method which is executed when a new instance of the class is created
- It has a special syntax

```
ClassName (parameters)
{
  // body
}
```

- that is, its name is the same as the class name, and
- it has no return type
- It is used to initialize the state of the object being created

Method types

```
public class Robot
{
    double x, y, direction;

    // Constructor
    Robot (double dir) { ... }

    // Mutators
    void turn(double angle) { ... }
    void advance(double distance) { ... }
    // Accessors
    double getX()
    {
        return x;
    }

    double getY()
    {
        return y;
    }
    double getDirection()
    {
        return direction;
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto, marc;
        ernesto = new Robot(Math.PI/2);
        marc = new Robot(0.0);
        ernesto.advance(200.0);
        marc.turn(Math.PI / 2);
        marc.advance(150.0);
        double d;
        d = marc.getDirection();
    }
}
```

Objects and Classes

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto, marc;
        ernesto = new Robot(Math.PI/2);
        marc = new Robot(0.0);
        ernesto.advance(200.0);
        marc.turn(Math.PI / 2);
        marc.advance(150.0);
        System.out.println( marc.getDirection() );
    }
}
```

Method calls in context

- There are two forms of method calls:
  - Method call as a statement
  - Method call as an expression

- A method call is a statement if its return type is `void`, otherwise it is an expression.

- If a method call is an expression, it must appear in a context that allows expressions, such as:

  A. the right hand-side of an assignment:

  ```
  double x;
  x = ernesto.getX();
  ```

  B. ...or, the argument of another method:

  ```
  System.out.println( marc.getDirection() );
  ernesto.turn( marc.getDirection() );
  ```

- But the types **must** match!

Methods are reusable abstractions

- A method is a procedure that you write once but you can invoke any number of times

- Suppose we want to have a robot perform the following task:

1. advance 10 units

2. turn right 90 degrees

3. advance 25 units

Methods are reusable abstractions

```
ernesto.advance(10.0);
ernesto.turn( -Math.PI / 2 );
ernesto.advance(25.0);
```

Methods are reusable abstractions

```
double ernesto_x, ernesto_y, ernesto_direction;
double dx, dy;
// initialize
ernesto_x = 0.0;
ernesto_y = 0.0;
ernesto_direction = 0.0;

// advance 10
dx = 10.0 * Math.cos(ernesto_direction);
dy = 10.0 * Math.sin(ernesto_direction);
ernesto_x = ernesto_x + dx;
ernesto_y = ernesto_y + dy;

// turn right
ernesto_direction = ernesto_direction
                    + -Math.PI / 2;
// advance 25
dx = 25.0 * Math.cos(ernesto_direction);
dy = 25.0 * Math.sin(ernesto_direction);
ernesto_x = ernesto_x + dx;
ernesto_y = ernesto_y + dy;
```

Methods

```java
// Brownian motion (drunk walk)
Robot ernesto;
Random generator = new Random();
double d, a;
a = generator.nextDouble() * Math.PI;
ernesto = new Robot(a);
while (true)
{
    d = generator.nextDouble() * 100.0;
    ernesto.advance(d);
    a = generator.nextDouble() * Math.PI - Math.PI / 2;
    ernesto.turn(a);
}
```

Scope

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

Scope

```
public class Dog
{
    void talk()
    {
        System.out.println(‘‘Woof! Woof!’’);
    }
}

public class Cat
{
    void talk()
    {
        System.out.println(‘‘Meowwww...’’);
    }
}
```

Scope

```
public class CatsAndDogs
{
    public static void main(String[] args)
    {
        Dog odie;
        Cat garfield;
        odie = new Dog();
        garfield = new Cat();
        odie.talk();
        garfield.talk();
    }
}
```

Scope

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

Scope

```
public class Dog
{
    String name;

    void talk()
    {
        System.out.println(‘‘Woof! Woof!’’);
    }
}

public class Cat
{
    String name;

    void talk()
    {
        System.out.println(‘‘Meowwww...’’);
    }
}
```

Scope

- The scope of a parameter of a method is only the method itself

- Therefore different mehods can have parameters with the same name

- A parameter exists in memory only while the method is being executed, and disappears when the method finishes.

Scope

```
public class BankAccount
{
    String owner;
    double balance;

    void withdraw(double amount)
    {
        balance = balance - amount;
    }

    void deposit(double amount)
    {
        balance = balance + amount;
    }
}
```

Scope

- The scope of a local variable in a method is only the method itself

- Therefore different mehods can have local variables with the same name

- A local variable exists in memory only while the method is being executed, and disappears when the method finishes.

Scope

```
public class Announcer
{
    void start()
    {
        String message;
        message = “Ready, set, go!”;
        System.out.println(message);
    }

    void stop()
    {
        String message;
        message = “...aaaaand STOP!”;
        System.out.println(message);
    }
}
```

Scope

```
public class Robot
{
    double x, y, direction;

    Robot (double dir)
    {
        x = 0.0;
        y = 0.0;
        direction = dir;
    }

    void turn(double angle)
    {
        direction = direction - angle;
    }

    // Continues below
    void advance(double distance)
    {
        double dx, dy;
        dx = distance * Math.cos(direction);
        dy = distance * Math.sin(direction);
        x = x + dx;
        y = y + dy;
    }
}  // End of Robot class
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        distance = 8;
    }
}
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        distance = 8;     // WRONG!
    }
}
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        direction = Math.PI;
    }
}
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        direction = Math.PI;    // WRONG!
    }
}
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        dy = 20.0;
    }
}
```

Scope

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(Math.PI/2);
        dy = 20.0;    // WRONG!
    }
}
```

Relations between objects

- A class may have attributes which are other objects
- ... therefore an object can have references to other objects
- An object can send a message to other objects
  - by calling methods from within its own methods

Objects can refer to other objects

- A class may have attributes which are other objects

```
public class Robot
{
    double x, y, direction;

    Robot (double dir) { ... }

    void turn(double angle) { ... }
    void advance(double distance) { ... }
}

public class BankAccount
{
    String owner;
    double balance;

    BankAccount (String who, double qty) { ... }
    void withdraw(double amount) { ... }
    void deposit(double amount) { ... }
}
```

Objects can refer to other objects

```
public class Robot
{
    double x, y, direction;
    BankAccount account;

    Robot (double dir) { ... }

    void turn(double angle) { ... }
    void advance(double distance) { ... }
}

public class BankAccount
{
    String owner;
    double balance;

    BankAccount (String who, double qty) { ... }
    void withdraw(double amount) { ... }
    void deposit(double amount) { ... }
}
```

Objects can refer to other objects

```
public class Robot
{
    double x, y, direction;
    BankAccount account;

    Robot (String name, double dir)
    {
        x = 0.0;
        y = 0.0;
        direction = dir;
        account = new BankAccount(name, 100.0);
    }

    void turn(double angle) { ... }
    void advance(double distance) { ... }
}
```

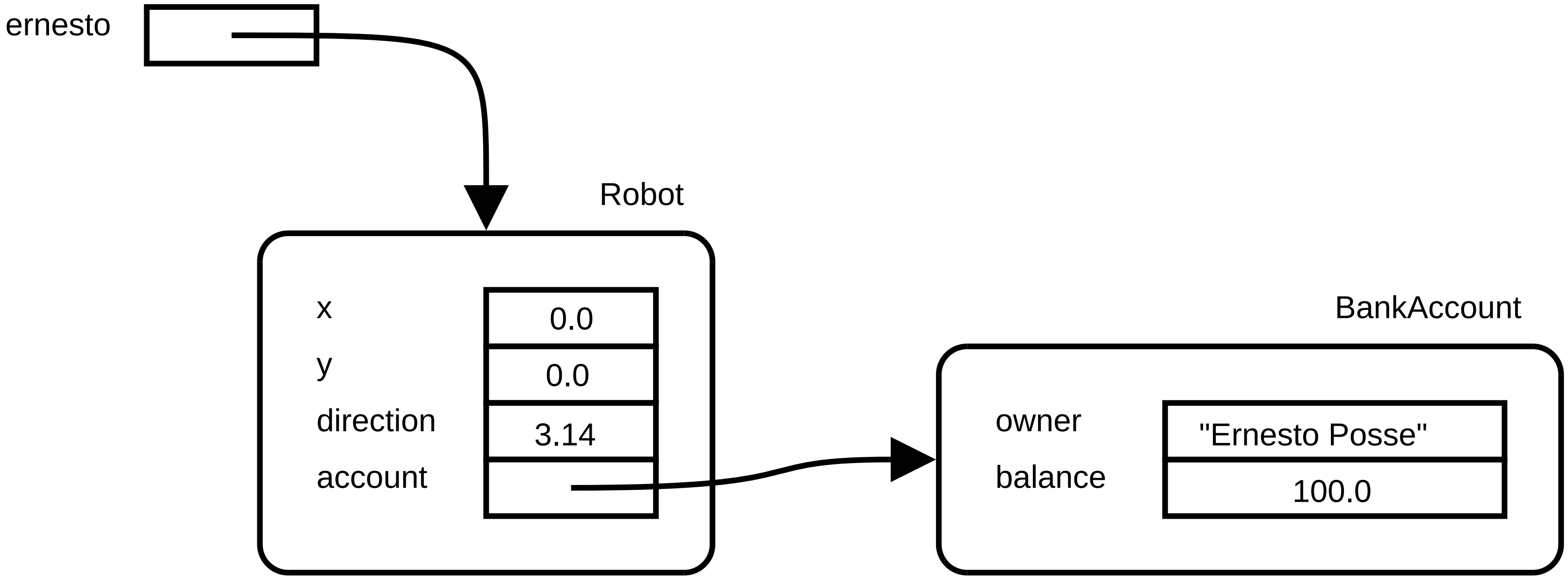
Objects can refer to other objects

```
public class Robot
{
    double x, y, direction;
    BankAccount account;
    double distance_covered;

    Robot (String name, double dir)
    {
        x = 0.0;
        y = 0.0;
        direction = dir;
        account = new BankAccount(name, 100.0);
        distance_covered = 0.0;
    }

    void turn(double angle) { ... }

    // continues below ...
    void advance(double distance)
    {
        double dx, dy;
        dx = distance * Math.cos(direction);
        dy = distance * Math.sin(direction);
        x = x + dx;
        y = y + dy;
        distance_covered = distance_covered
                           + distance;
        if (distance_covered >= 100.0)
        {
            account.deposit(50.0);
            distance_covered = 0.0;
        }
    }
}
```

Objects can refer to other objects

```
public class Test
{
    public static void main(String[] args)
    {
        Robot ernesto;
        ernesto = new Robot(“Ernesto Posse”, Math.PI/2);
        ernesto.advance(60.0);
        ernesto.advance(20.0);
        ernesto.advance(30.0);
    }
}
```

Objects can refer to other objects



Objects can refer to other objects

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
    distance_covered = distance_covered
                         + distance;
    if (distance_covered % 100 == 0)
    {
        account.deposit(50.0);
    }
}
```

The “this” reference

- `this` is a special variable that refers to tha object executing a method

- it can be used by an object to send a message to itself

The "this" reference

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
    distance_covered = distance_covered
                       + distance;
    if (distance_covered % 100 == 0)
    {
        account.deposit(50.0);
    }
}
```

The "this" reference

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(this.direction);
    dy = distance * Math.sin(this.direction);
    this.x = this.x + dx;
    this.y = this.y + dy;
    this.distance_covered = this.distance_covered
                       + distance;
    if (this.distance_covered % 100 == 0)
    {
        this.account.deposit(50.0);
    }
}
```

The "this" reference

- Sending a message to self

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
    distance_covered = distance_covered
                       + distance;
    if (distance_covered % 100 == 0)
    {
        account.deposit(50.0);
        this.turn(Math.PI);
    }
}
```

The "this" reference

- Sending a message to self

```
void advance(double distance)
{
    double dx, dy;
    dx = distance * Math.cos(direction);
    dy = distance * Math.sin(direction);
    x = x + dx;
    y = y + dy;
    distance_covered = distance_covered
                       + distance;
    if (distance_covered % 100 == 0)
    {
        account.deposit(50.0);
        turn(Math.PI);
    }
}
```

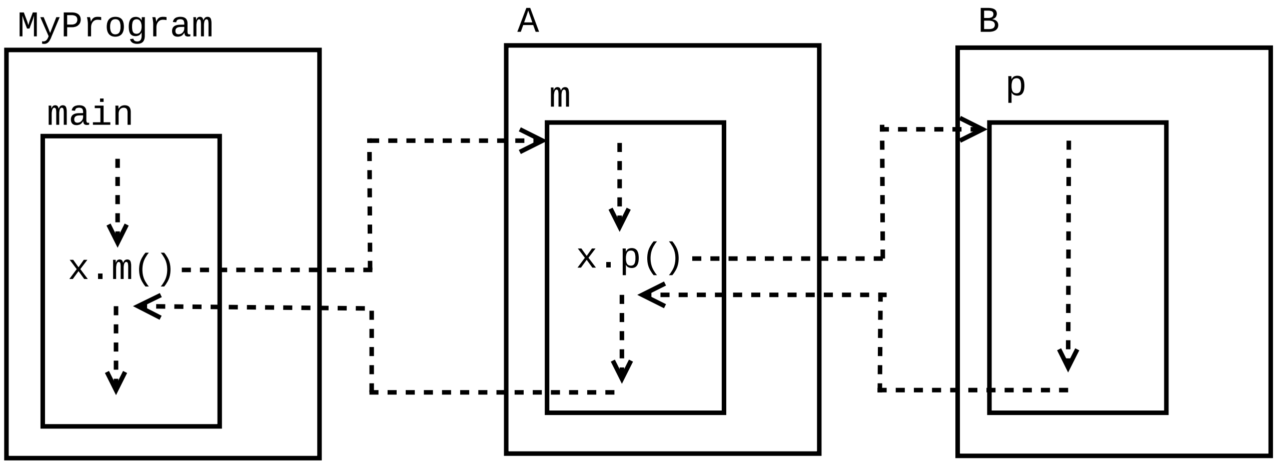
Method invocation

- Whenever a method is invoked two things happen:
  - Control flow: execution jumps from the current statement to the corresponding method
  - Data flow:
    - information is passed to the method as arguments
    - the method may return information to the caller

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”);
  }
}
```

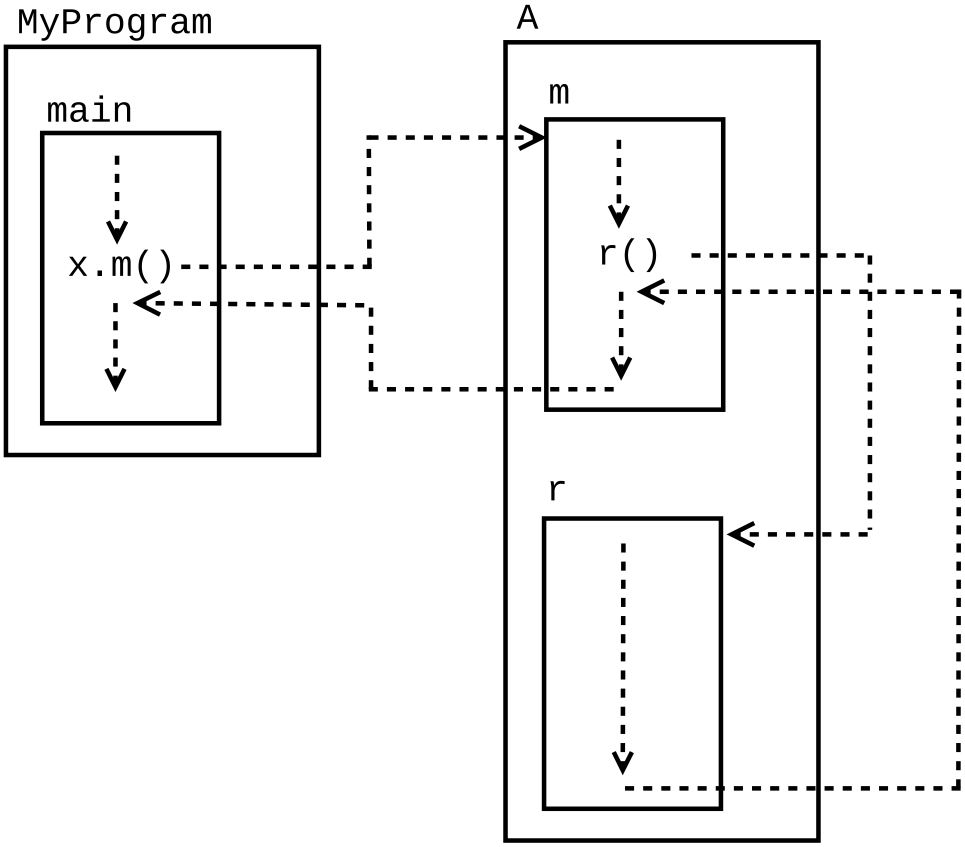
Method invocation: control flow



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: control flow

```
public class X {
  void f()
  {
    System.out.println(1);
  }
}
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.f();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
1
3
```

Method invocation: control flow

```
public class X {
  void f()
  {
    System.out.println(1);
  }
}
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.f();
    obj1.f();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
1
1
3
```

Method invocation: control flow

```java
public class X {
  void f()
  {
    System.out.println(1);
  }
  void g()
  {
    System.out.println(4);
  }
}
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.f();
    obj1.g();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
1
4
3
```

Method invocation: control flow

```java
public class X {
  void f()
  {
    System.out.println(1);
  }
  void g()
  {
    System.out.println(4);
  }
}
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.g();
    obj1.f();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
4
1
3
```

Method invocation: control flow

```
public class X {
  void g()
  {
    System.out.println(4);
  }
  void f()
  {
    System.out.println(1);
  }
}
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.f();
    obj1.g();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

  ```
  2
  1
  4
  3
  ```

- The order in which methods are declared in a class does not matter, but the order in which they are invoked does matter

Method invocation: control flow

```
public class X {
  void f()
  {
    System.out.println(1);
  }
}
public class Y {
  void g()
  {
    System.out.println(4);
  }
}
```

Method invocation: control flow

```
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    System.out.println(2);
    obj1.f();
    obj1.g();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints nothing! It is a compile-time error: `g` is not defined in class `X` which is the type of `obj1`

- A method can be applied to an object only if it is defined in the object's class.

Method invocation: control flow

```
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    Y obj2 = new Y();
    System.out.println(2);
    obj1.f();
    obj2.g();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
1
4
3
```

Method invocation: control flow

```
public class X {
  void f()
  {
    System.out.println(1);
  }
}
public class Y {
  void g()
  {
    X obj3 = new X();
    System.out.println(4);
    obj3.f();
    System.out.println(5);
  }
}
```

Method invocation: control flow

```
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    Y obj2 = new Y();
    System.out.println(2);
    obj1.f();
    obj2.g();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

```
2
1
4
1
5
3
```

Method invocation: control flow

```
public class X {
  void f()
  {
    System.out.println(1);
  }
}
public class Y {
  void f()
  {
    System.out.println(4);
  }
}
```

Method invocation: control flow

```
public class Z {
  public static void main(String[] args)
  {
    X obj1 = new X();
    Y obj2 = new Y();
    System.out.println(2);
    obj1.f();
    obj2.f();
    System.out.println(3);
  }
}
```

Method invocation: control flow

- Prints

  ```
  2
  1
  4
  3
  ```

- Different classes can have methods with the same name!

- Different classes can have attributes with the same name!

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.

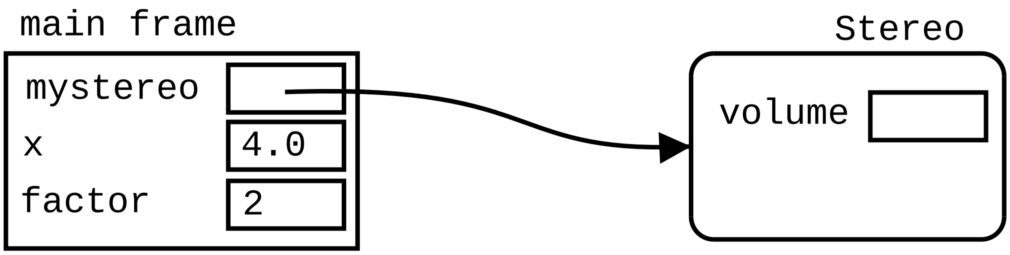
Fifth: when the end of the method is reached, or a `return` statement is reached, stop the method, the frame is discarded, and return to the calling method. The calling method is then resumed in the instruction immediatly after the method call.

Method invocation: Example

```
public class Stereo {
    double volume;
    void set_volume(double v)
    {
        volume = v;
    }
    double get_volume()
    {
        return volume;
    }
}
public class SoundSystemTest {
    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

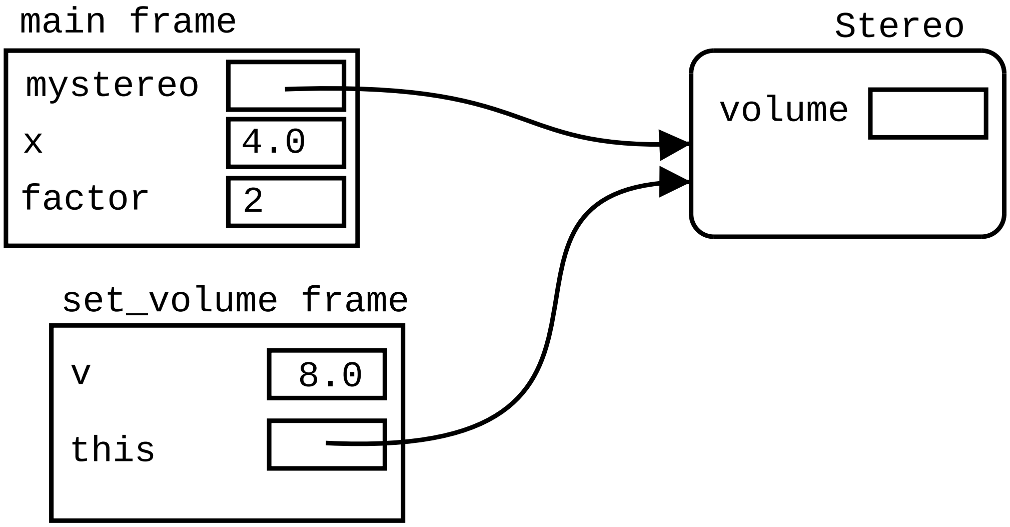
Before calling `mystereo.set_volume(x*factor)`



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;`



main frame

mystereo

x 4.0

factor 2

Stereo

volume

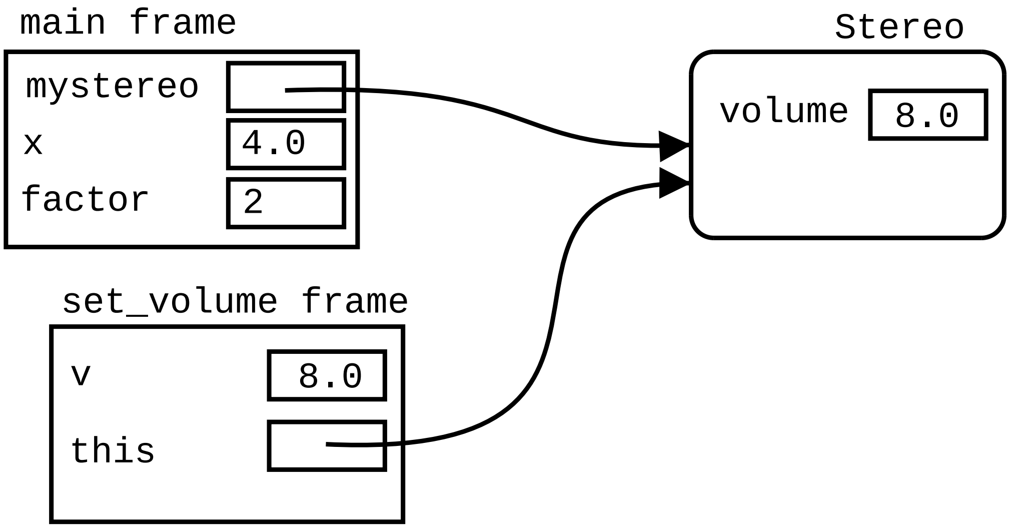
set_volume frame

v 8.0

this

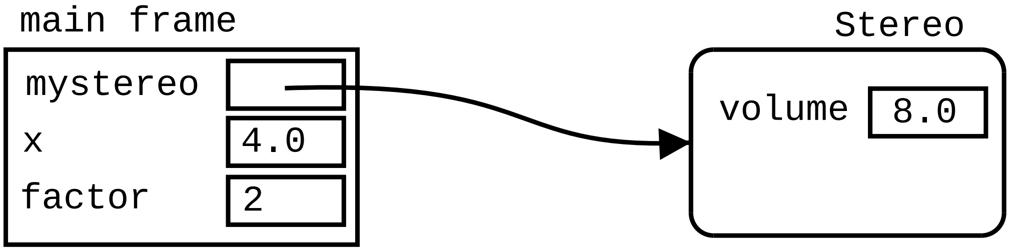
Method invocation: Memory structure

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.



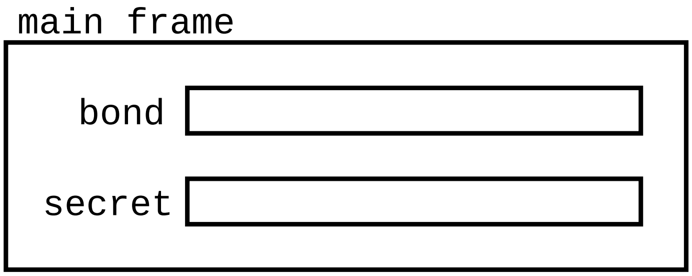
Method invocation

```
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;
  }

  //continues below...
  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’’);
  }
}
```

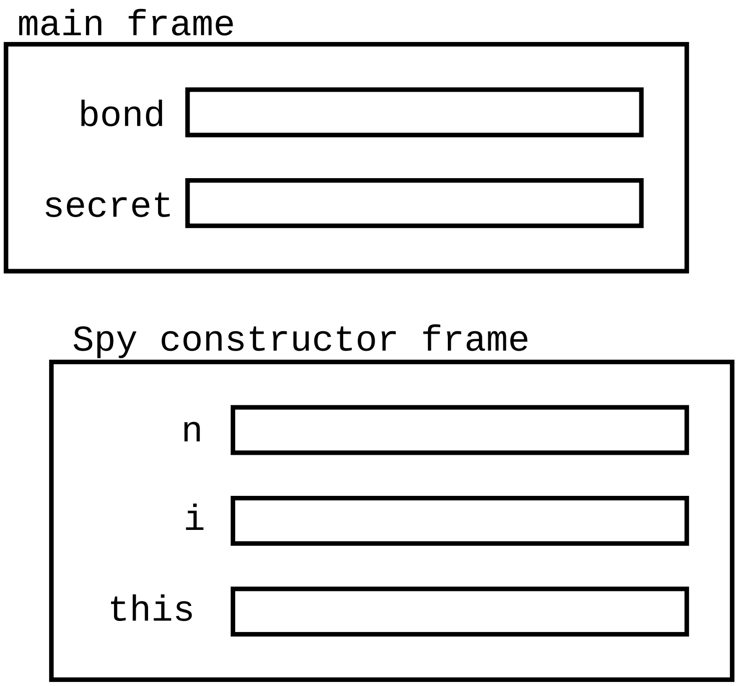
Method invocation



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



main frame

bond

secret

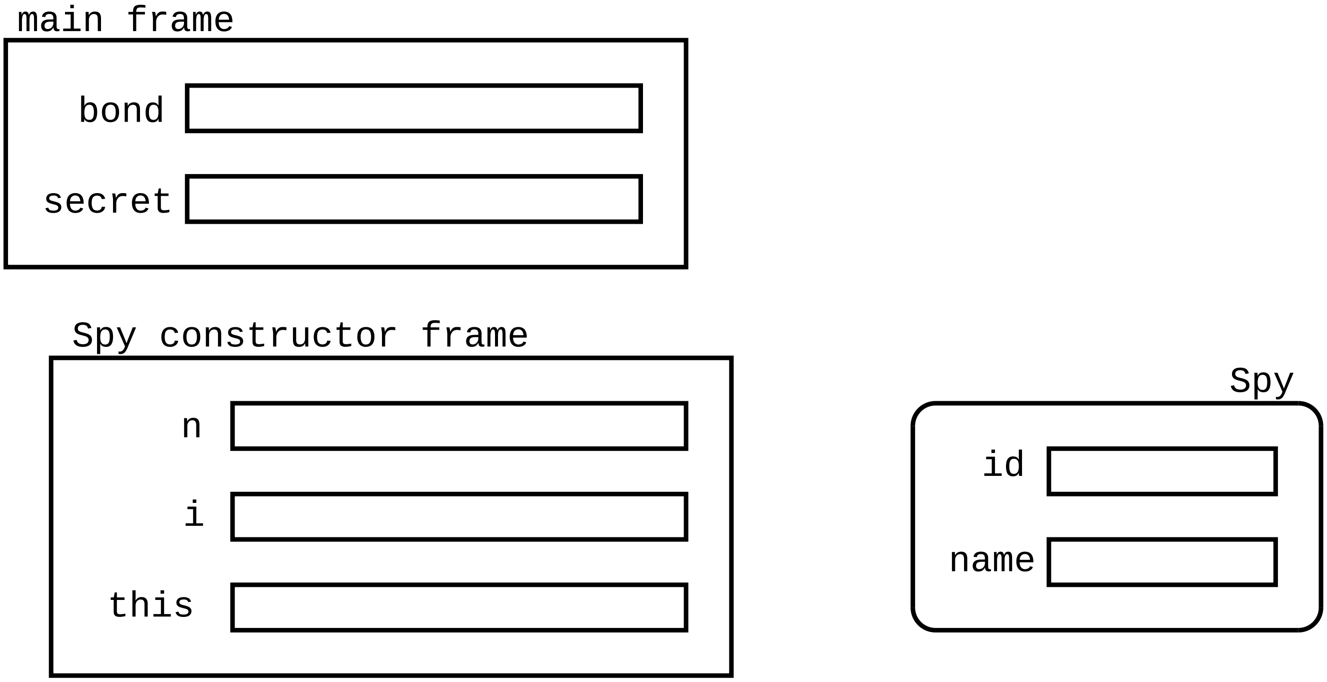
Spy constructor frame

n

i

this

Method invocation



main frame

bond

secret

Spy constructor frame

n

i

this

Spy

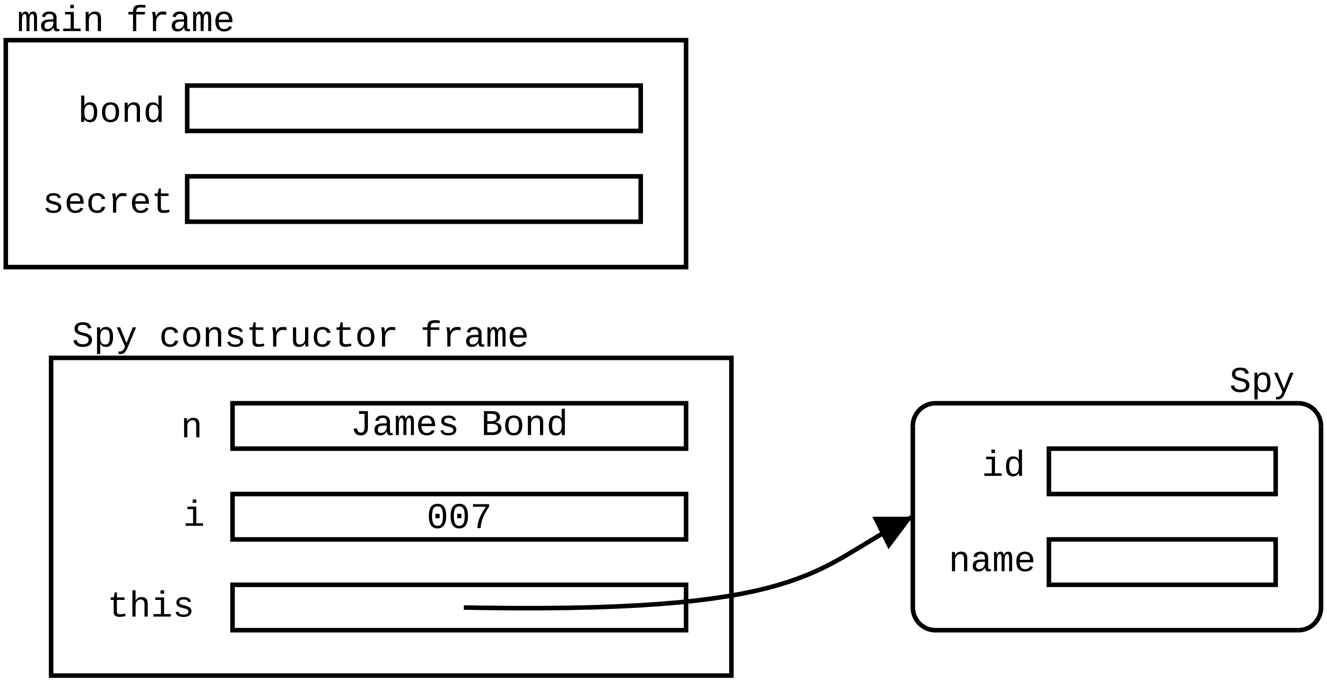
id

name

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;
  }
}
```

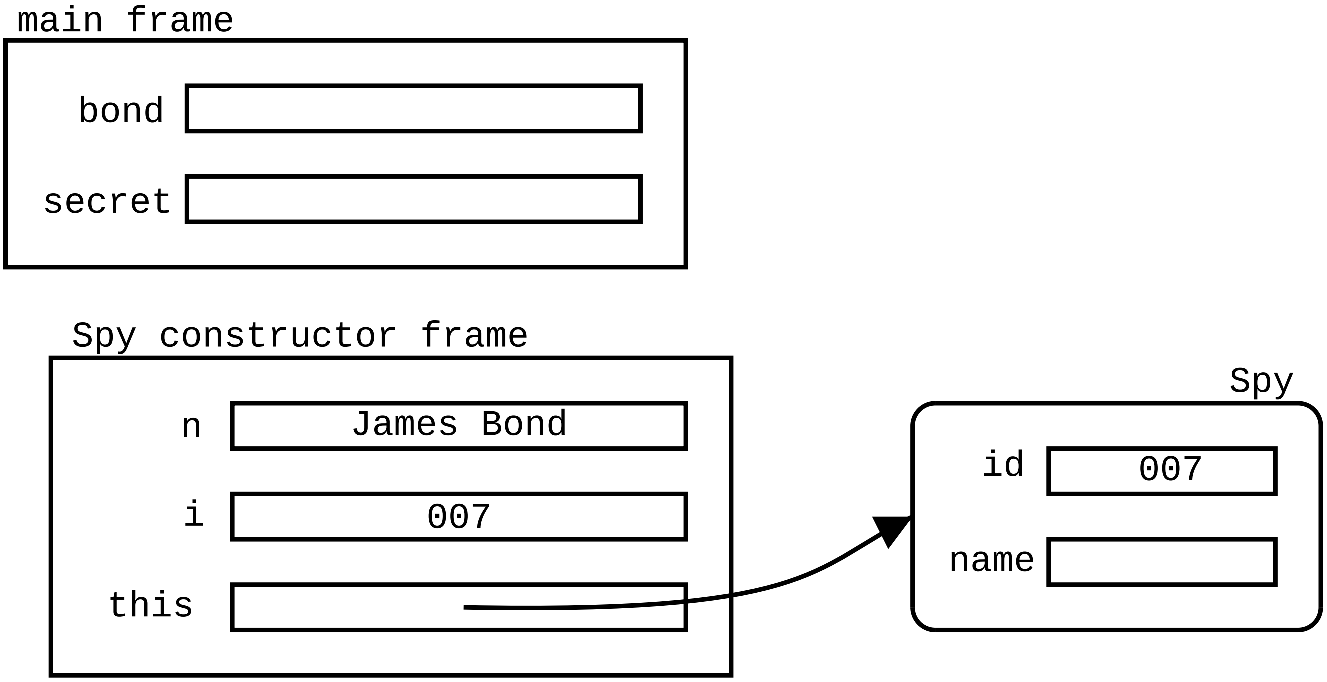
Method invocation



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;
  }
}
```

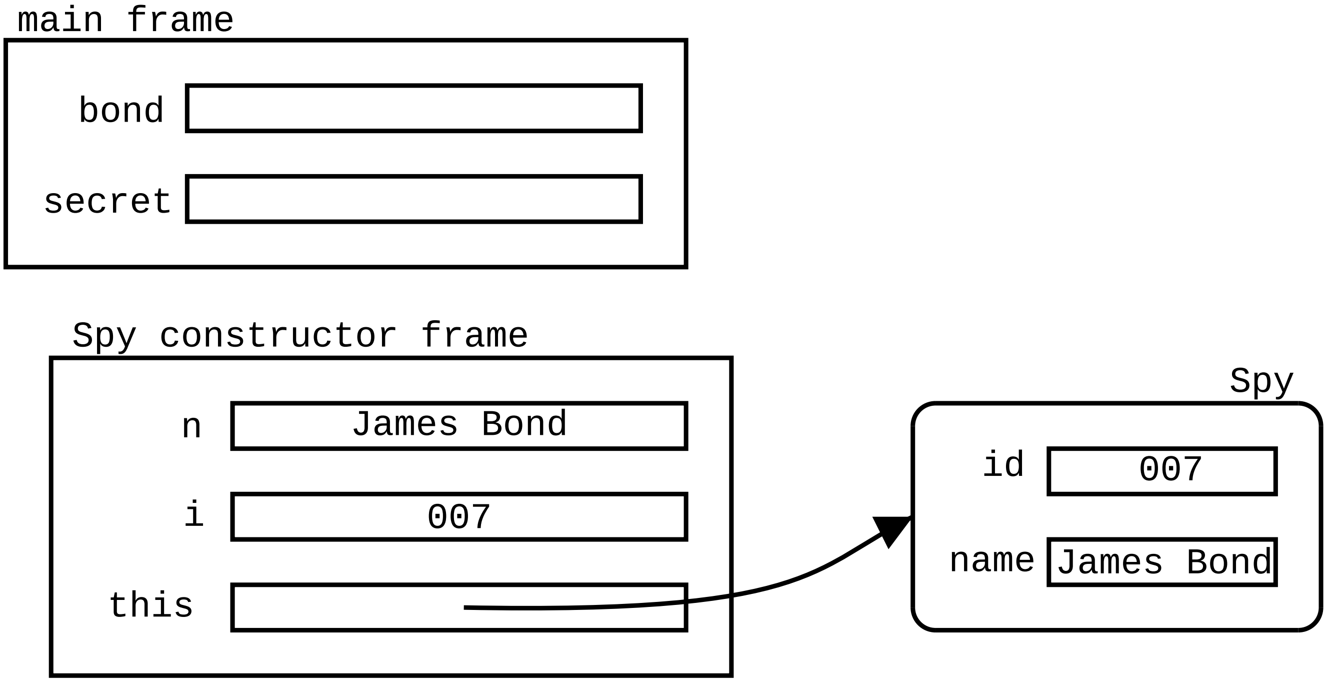
Method invocation



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



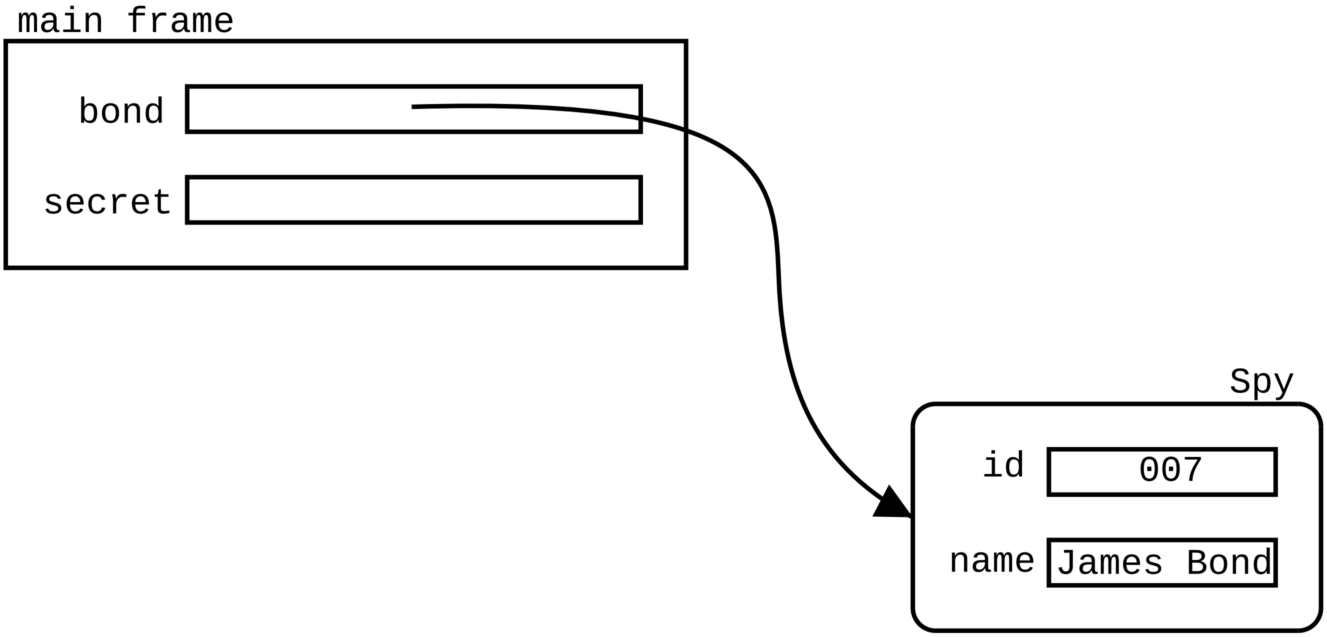
Method invocation

```
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”);
  }
}
```

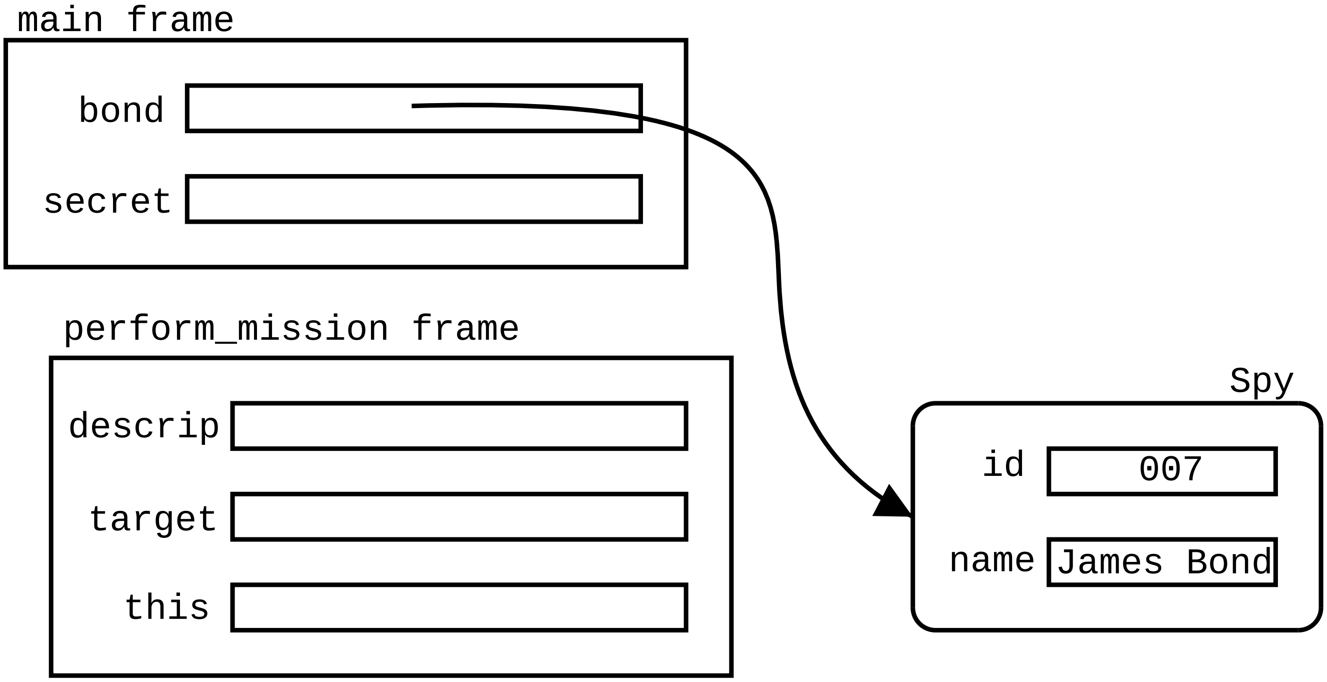
Method invocation



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”);
  }
}
```

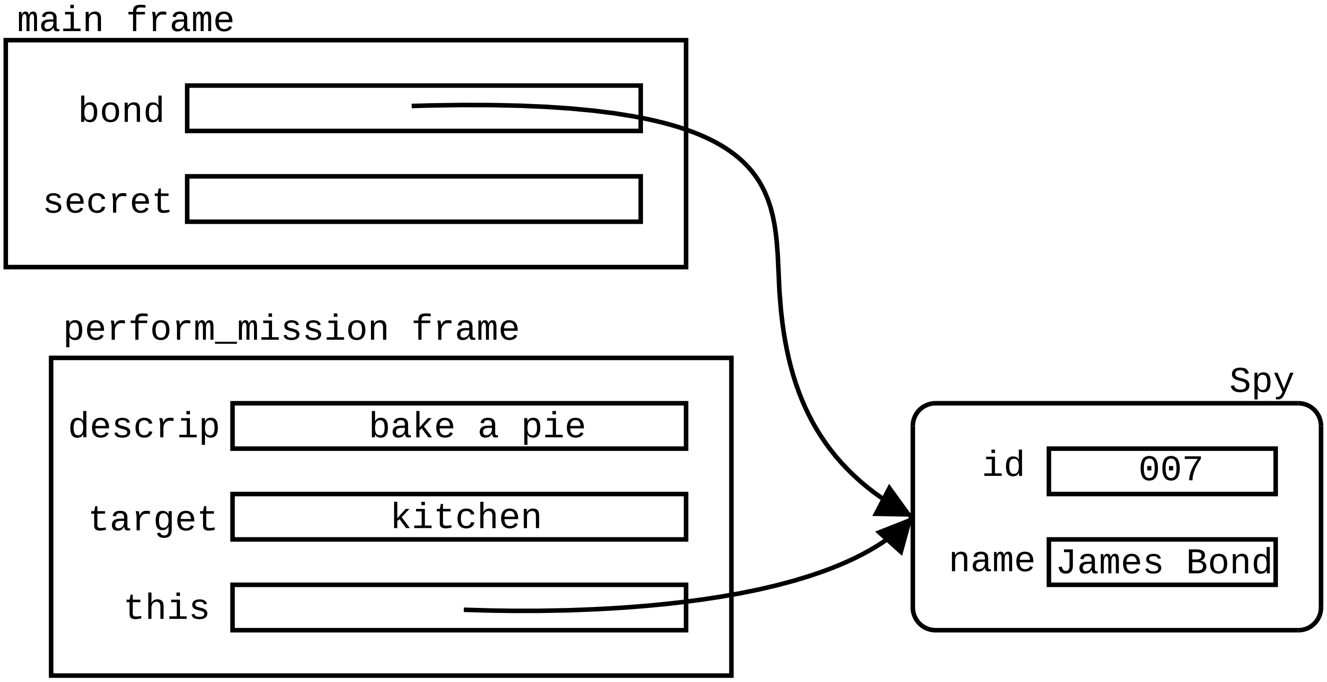
Method invocation



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;
  }
}
```

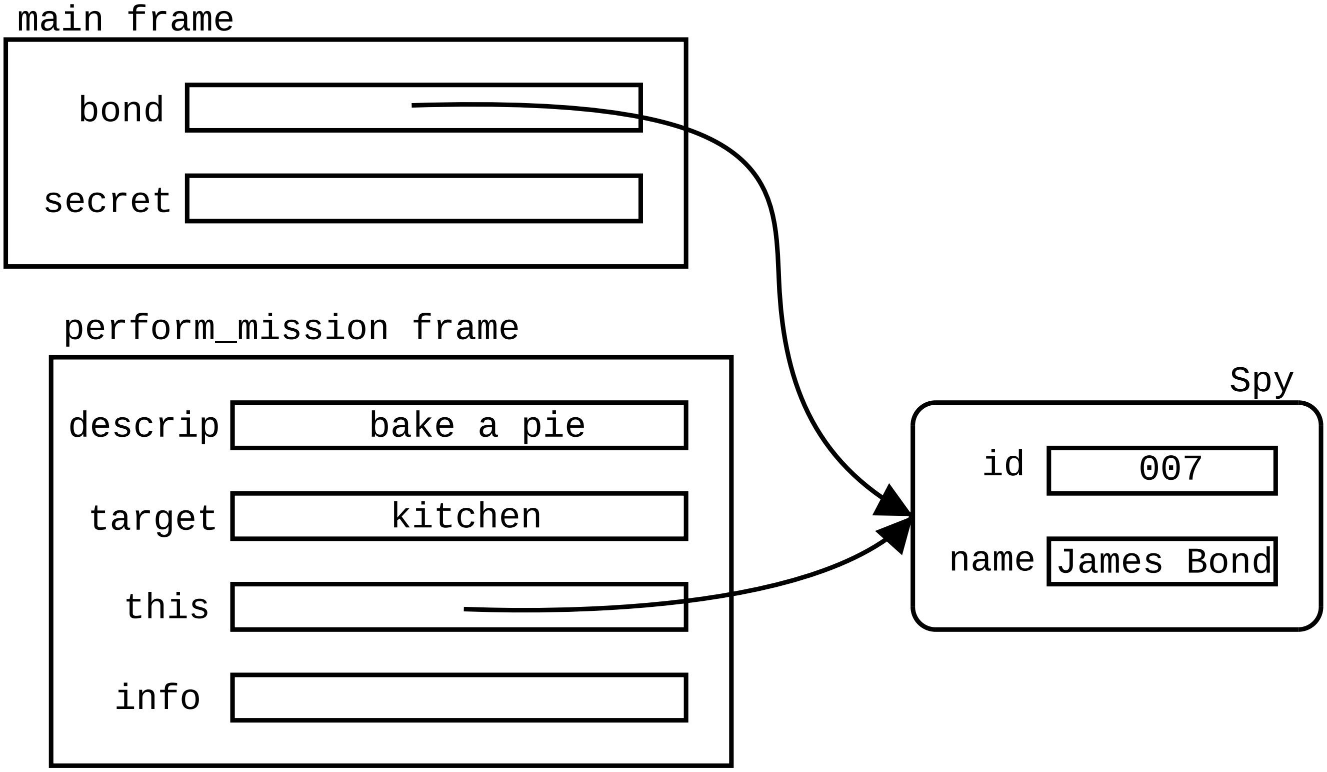
Method invocation



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;
  }
}
```

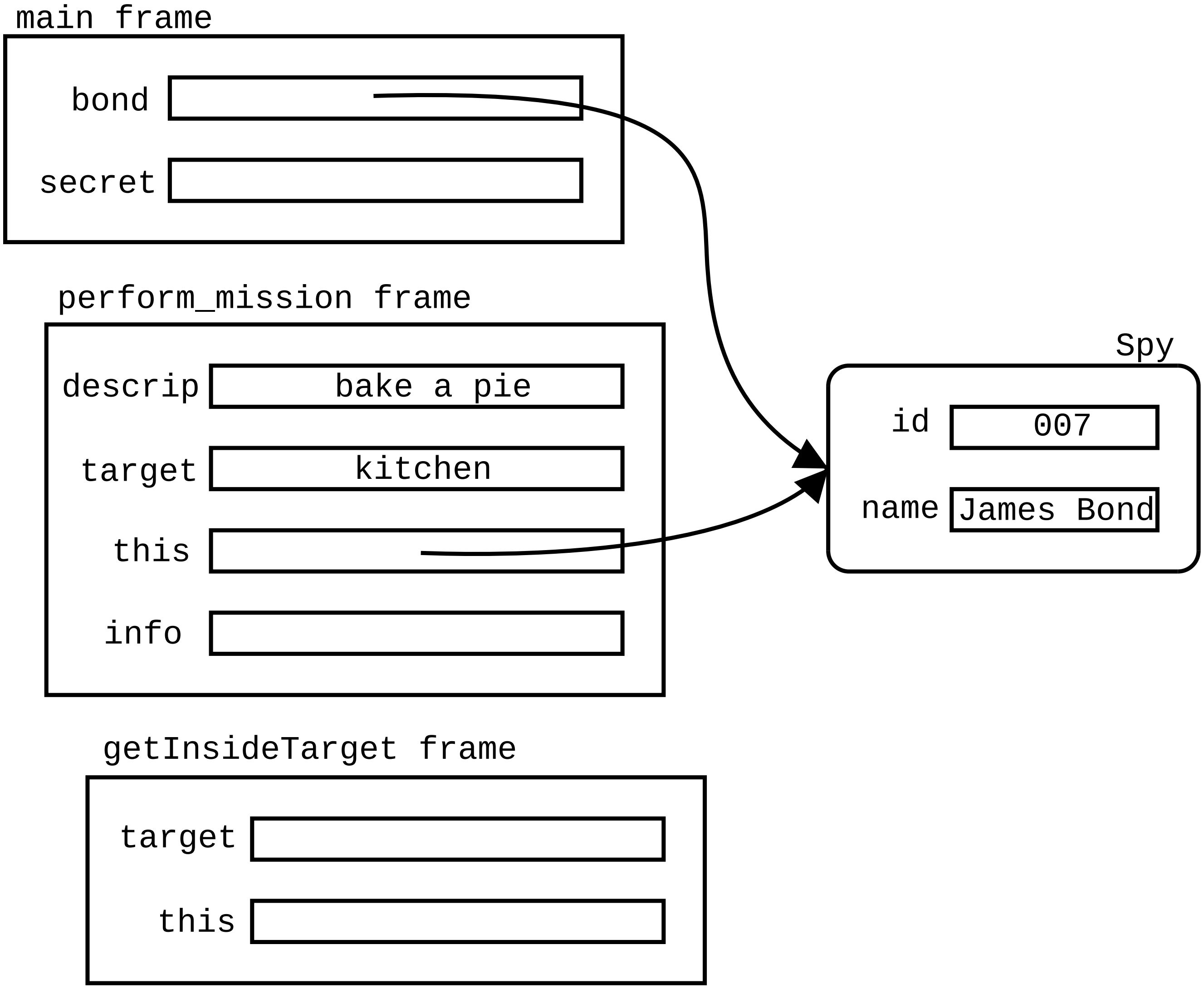
Method invocation



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;
  }
}
```

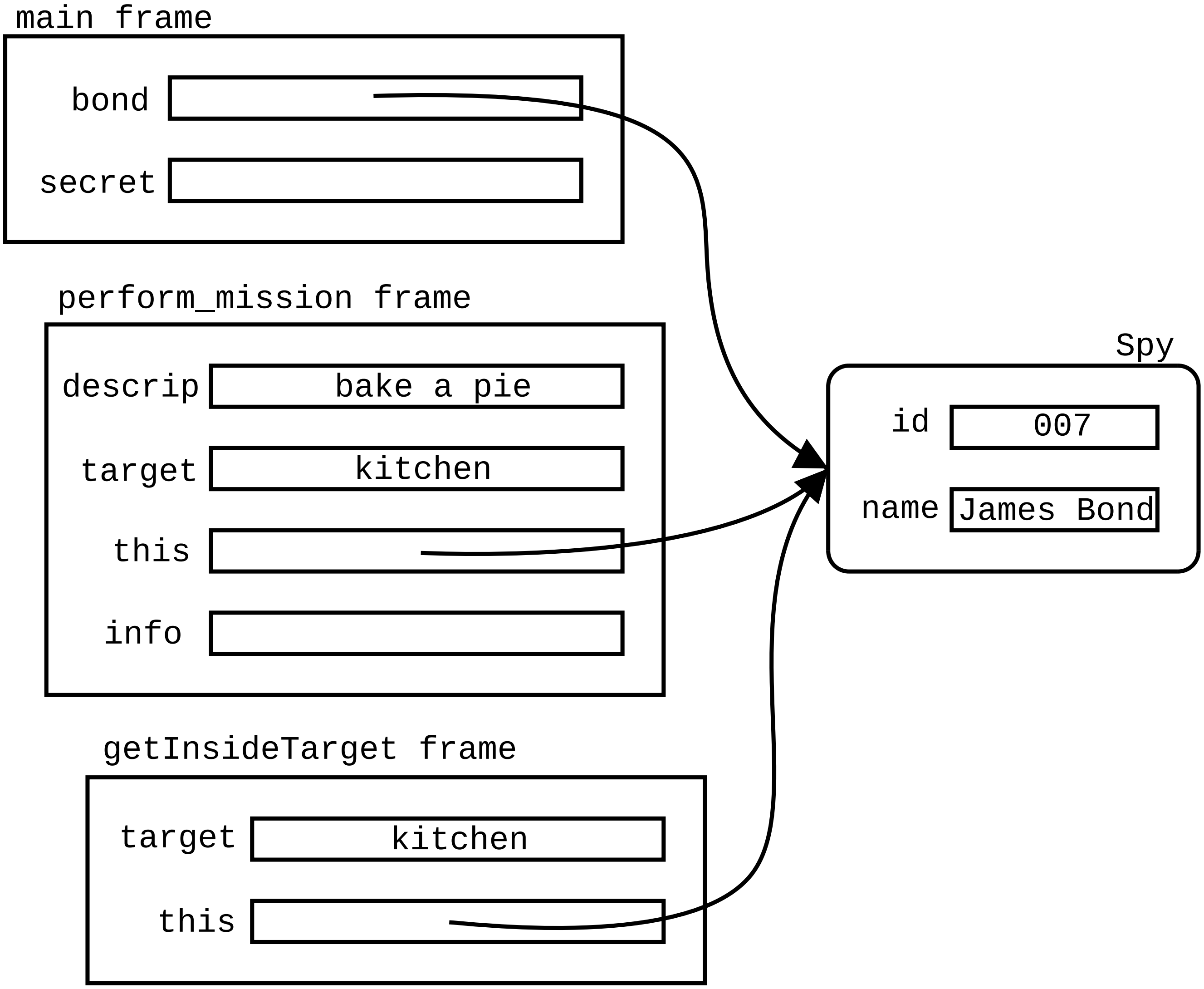
Method invocation



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;
  }
}
```

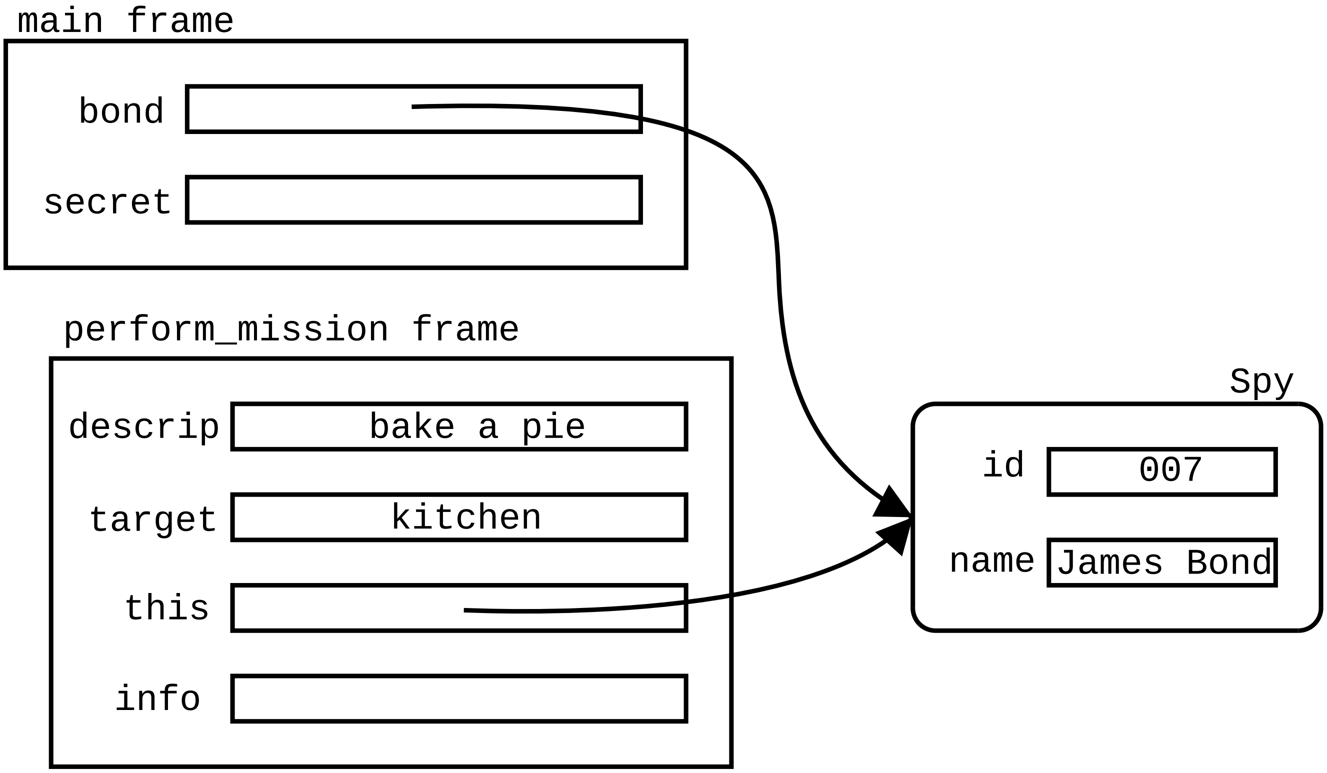
Method invocation

```
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 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;
  }
}
```

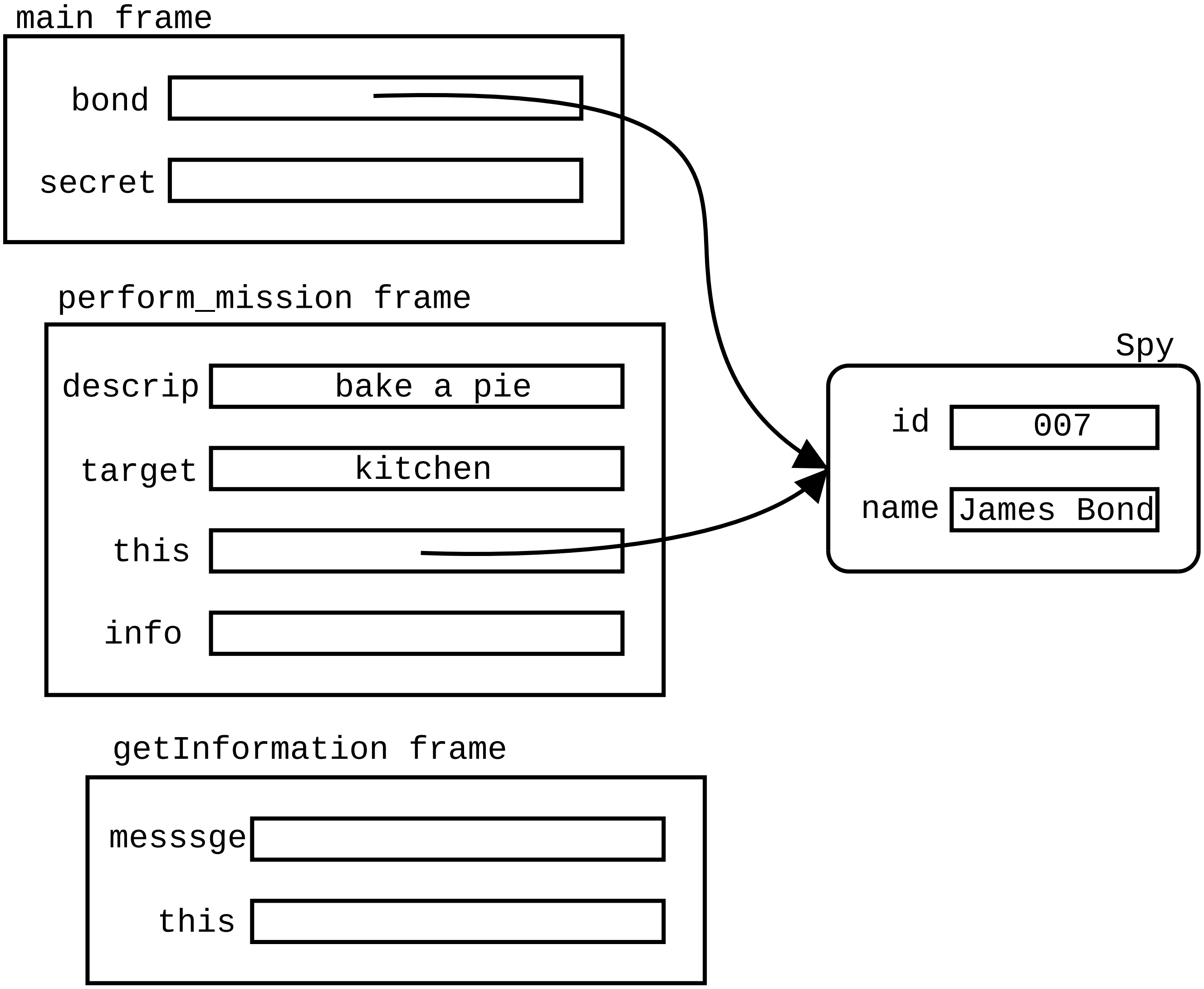
Method invocation



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



main frame

bond

secret

perform_mission frame

descrip | bake a pie

target | kitchen

this

info

getInformation frame

messsge

this

Spy

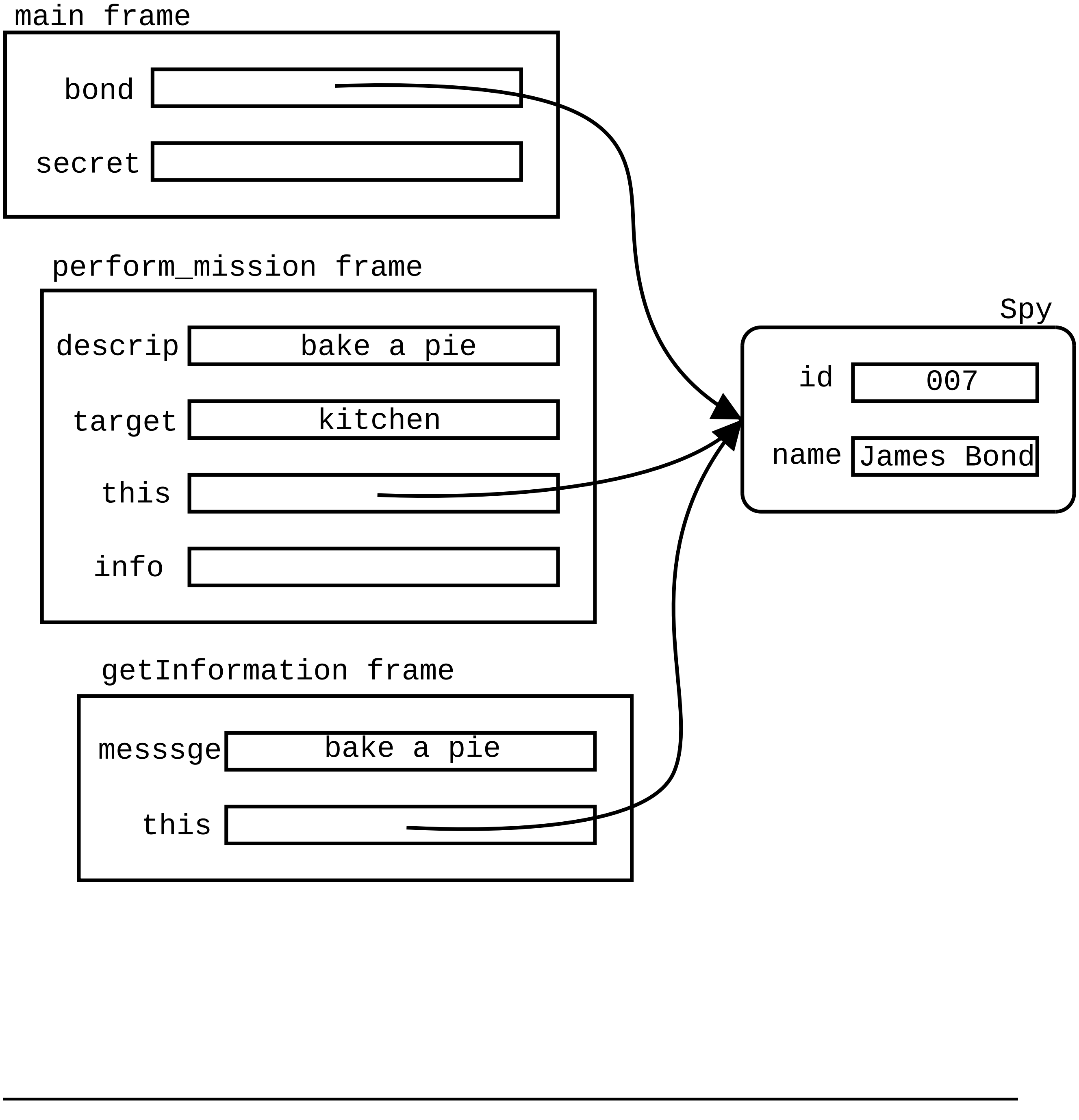
id | 007

name | James Bond

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;
  }
}
```

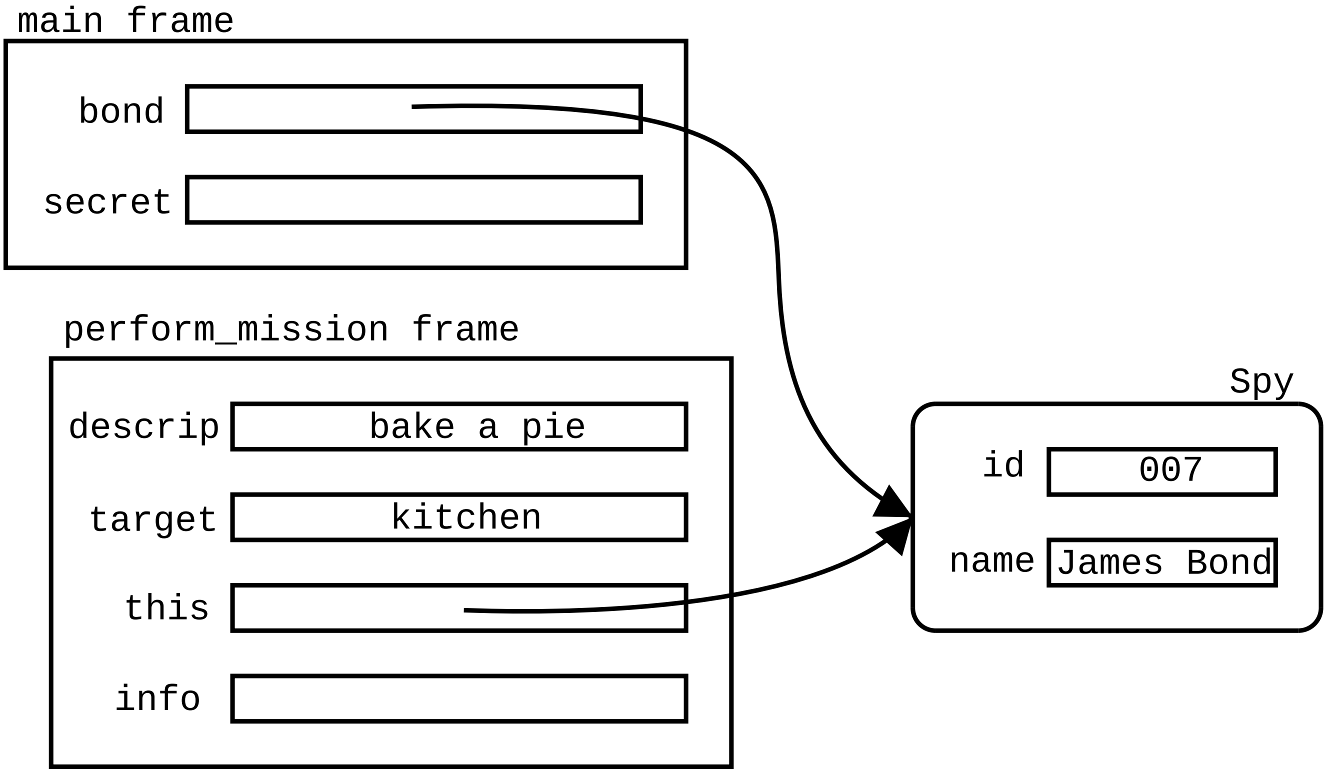
Method invocation



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;
  }
}
```

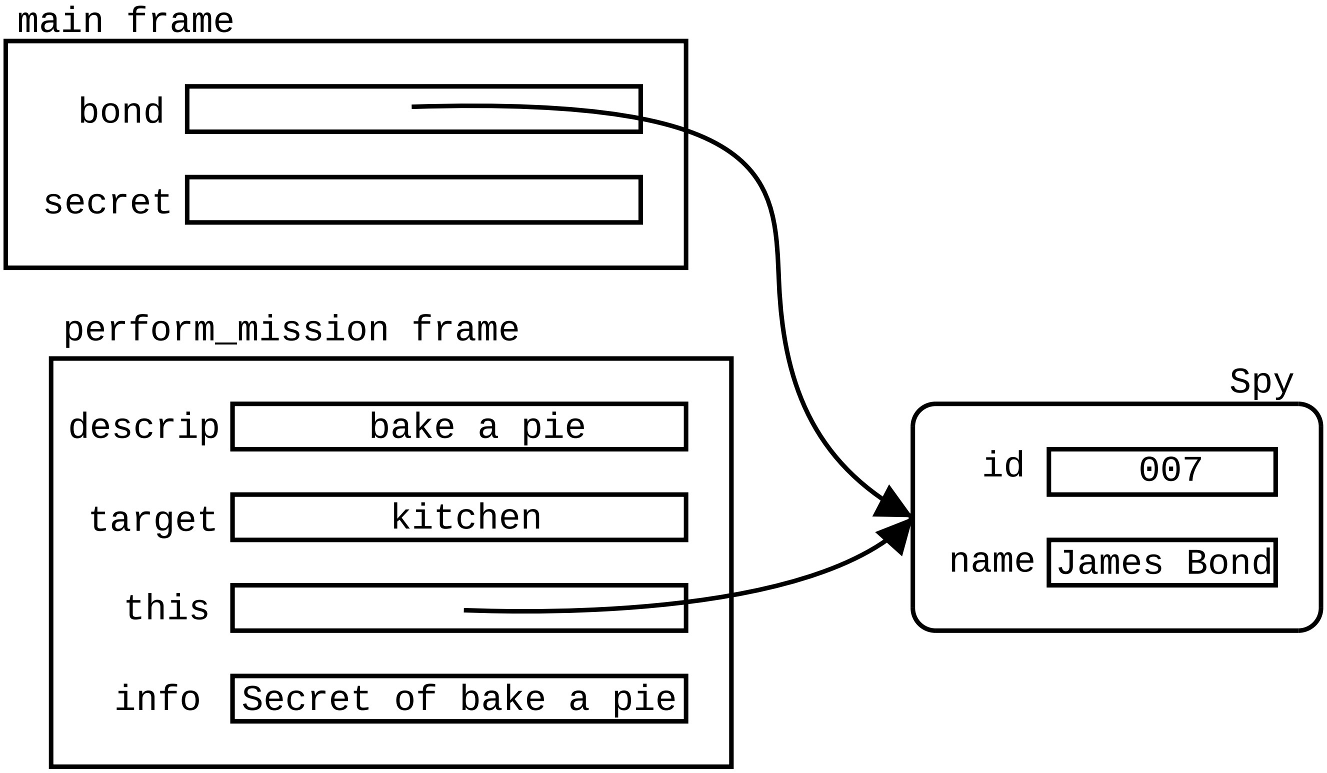
Method invocation



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;
  }
}
```

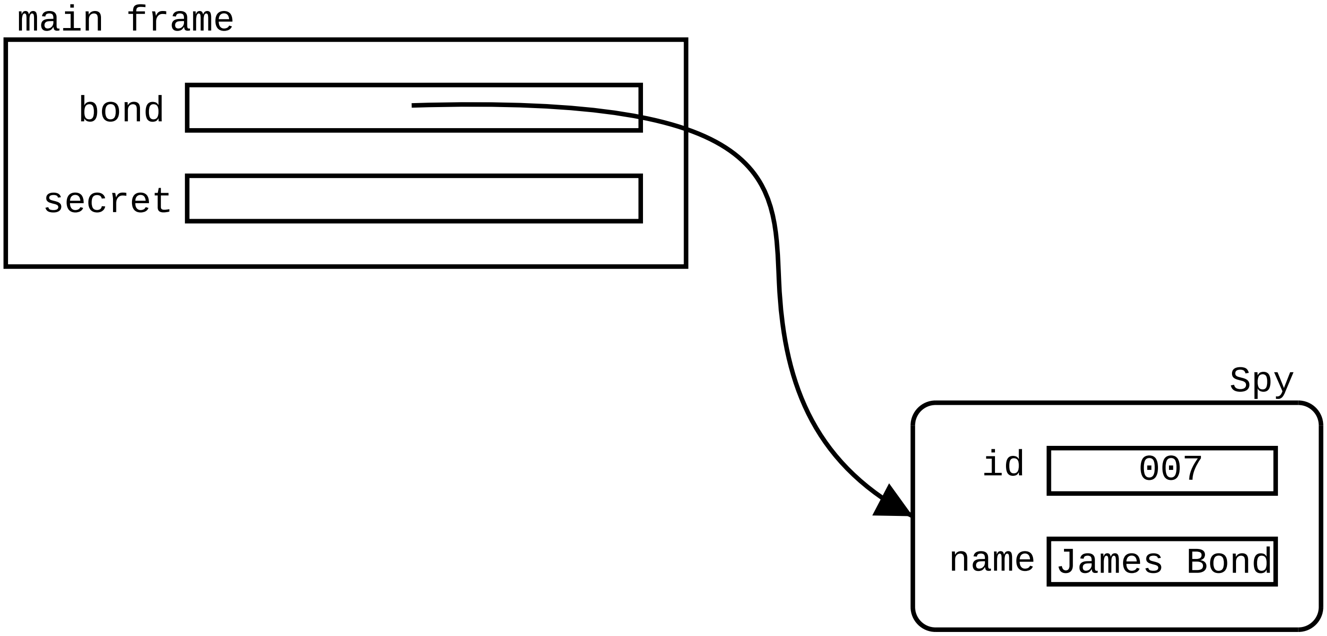
Method invocation



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;
  }
}
```

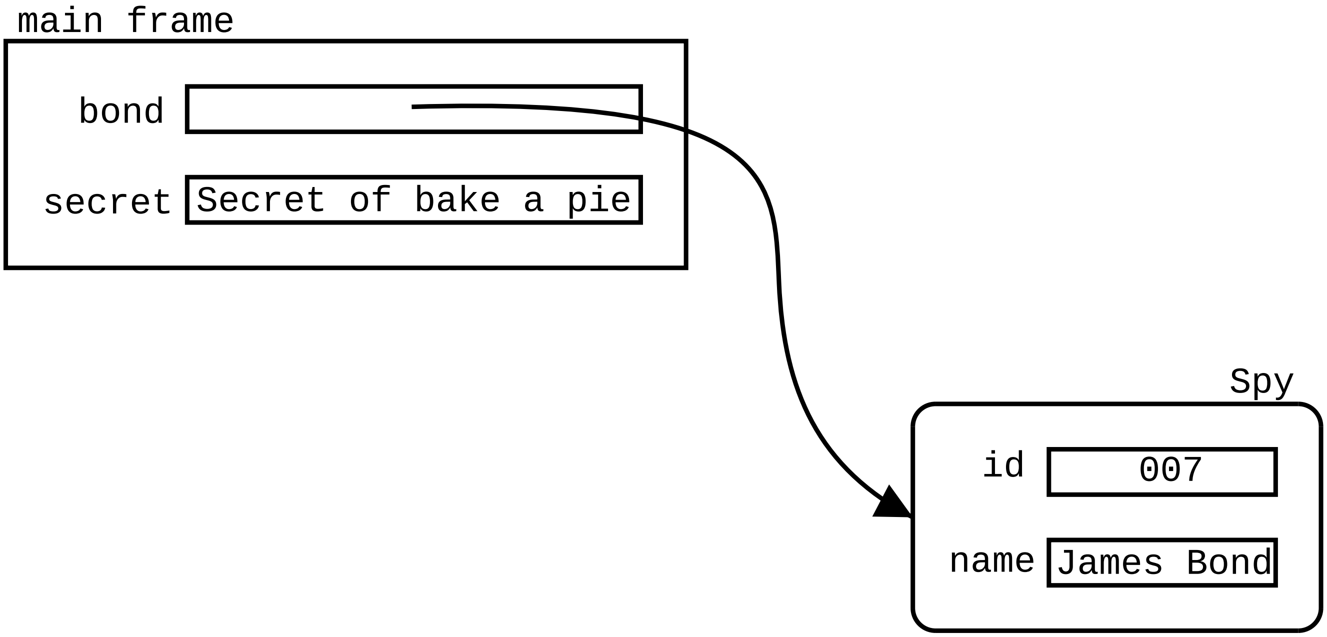
Method invocation



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



The end