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

```
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

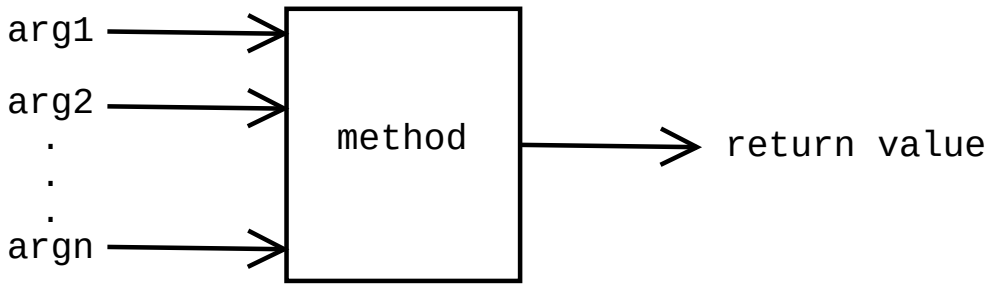
- Each object has its own separate *identity*, its own individual *state*
- The *state* of an object is the current value of its attributes
- The state of an object can change:
  - when we ask the object to do something
  - ... therefore, the methods of the object's class are responsible for changes to the object's state

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

```
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() );
    }
}
```

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

- A variable is *in scope* at some point in the program, if it is directly accessible

- In a method the following variables are accessible:
  - The method's parameters
  - The method's local variables
  - The class's attributes

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
```

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);
    }
}
```

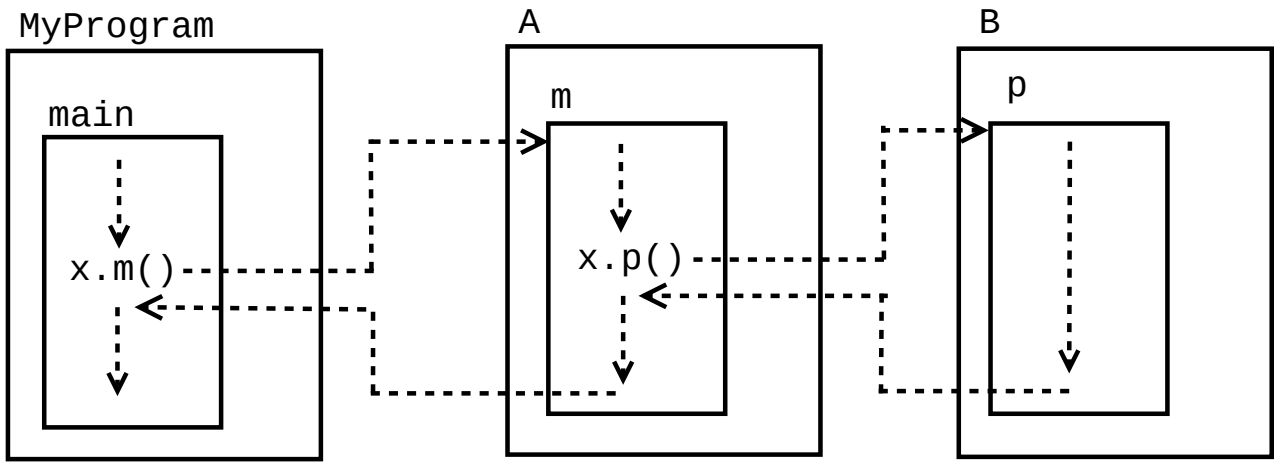
Method invocation

- Whenever a method is invoked two things happen:
  - Control flow: Method invocation: parameter passing 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: 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.

Example

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    title = t;
    director = d;
  }
  void print()
  {
    System.out.println(title);
    System.out.println(director);
  }
}
```

Is the same as...

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

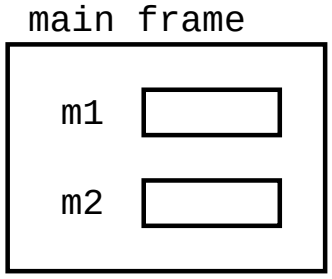
Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

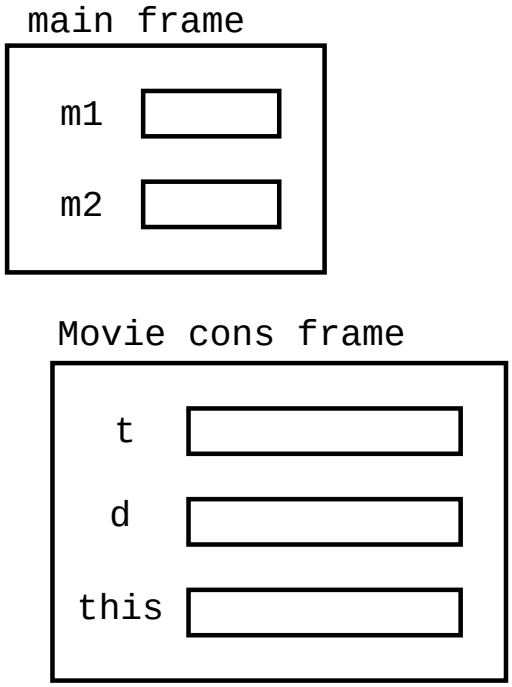
Method invocation: parameter passing



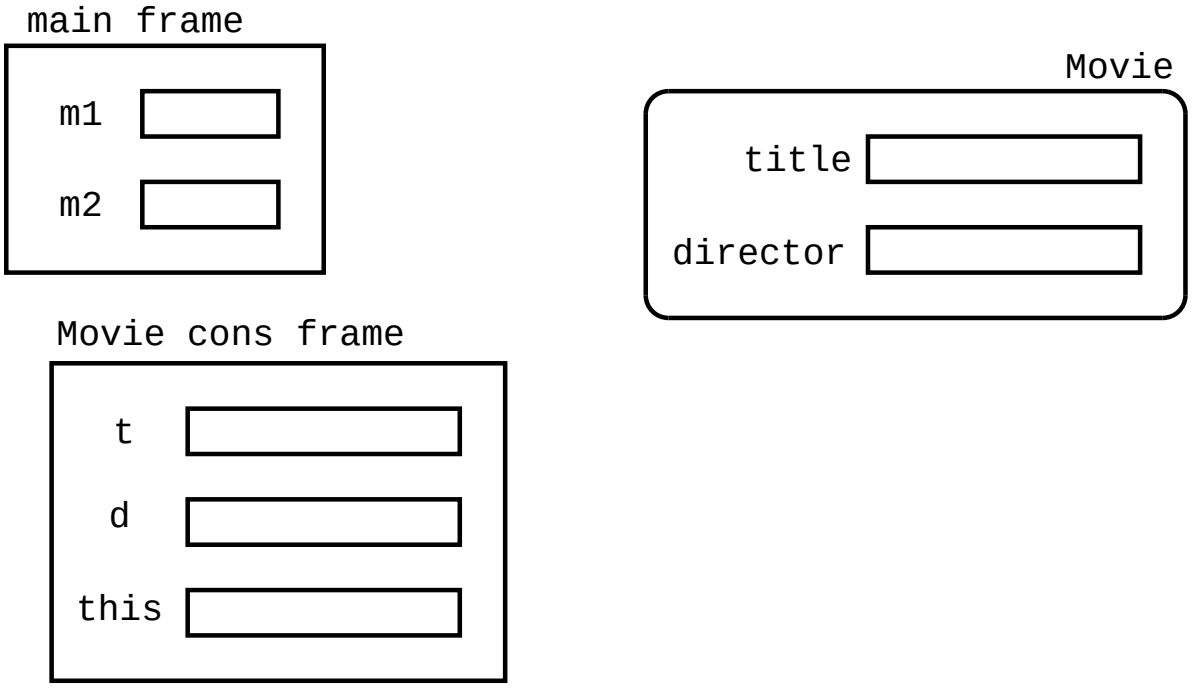
Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

Method invocation: parameter passing



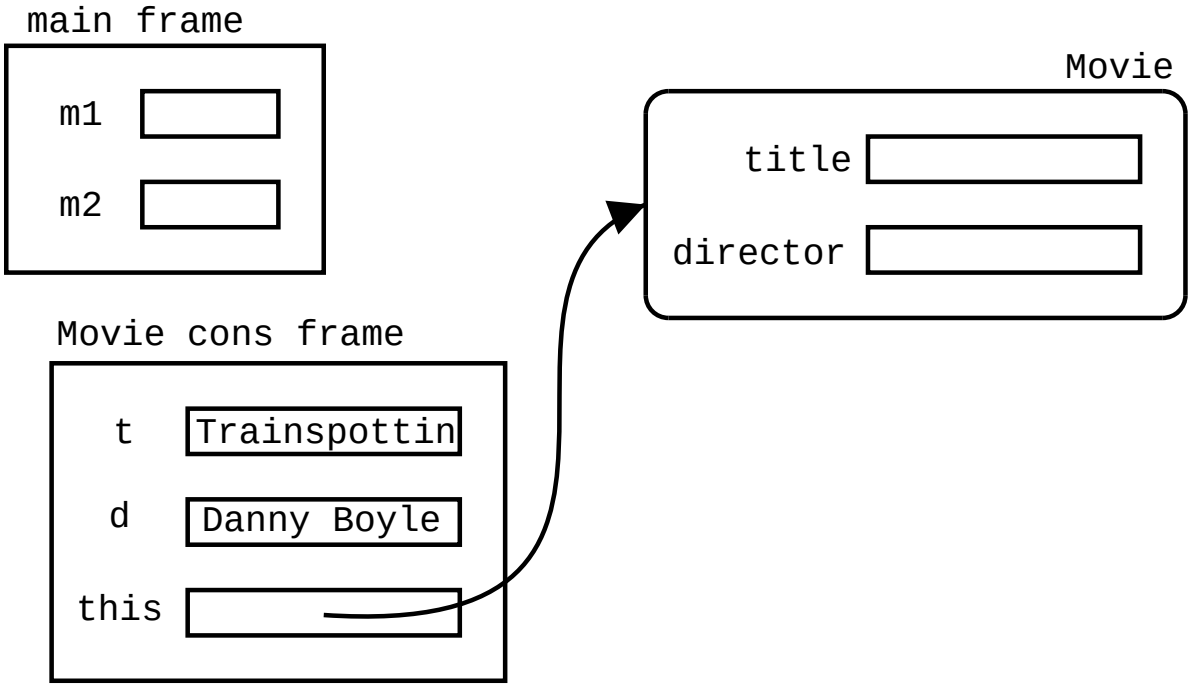
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

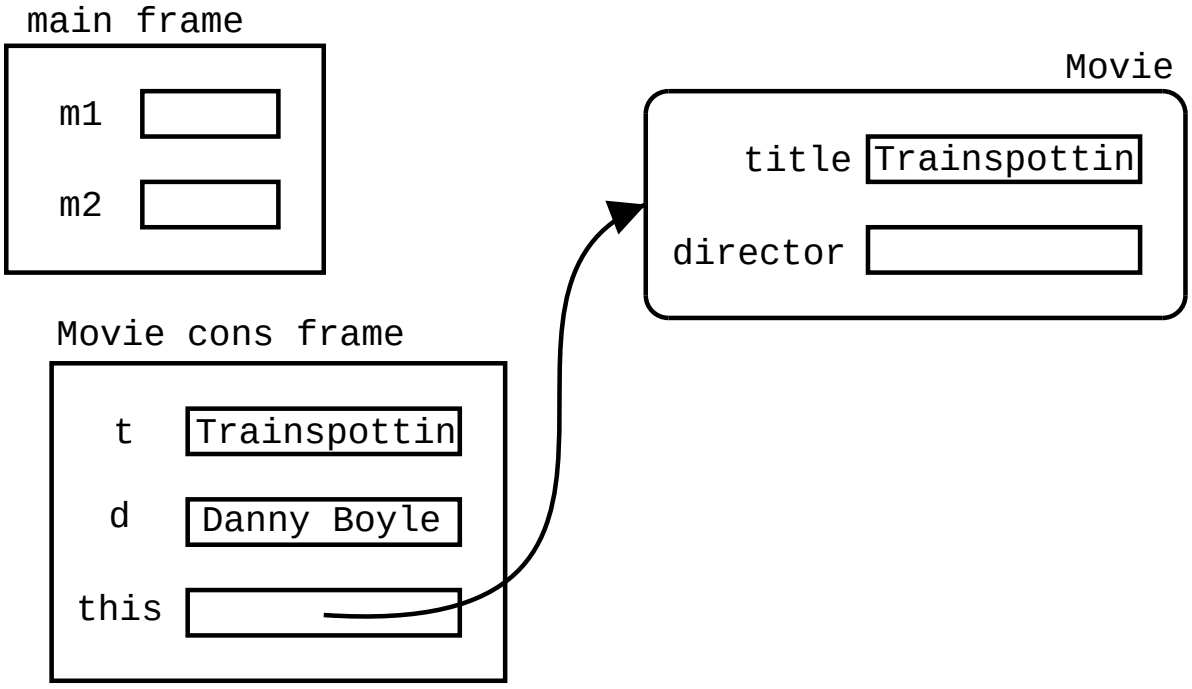
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

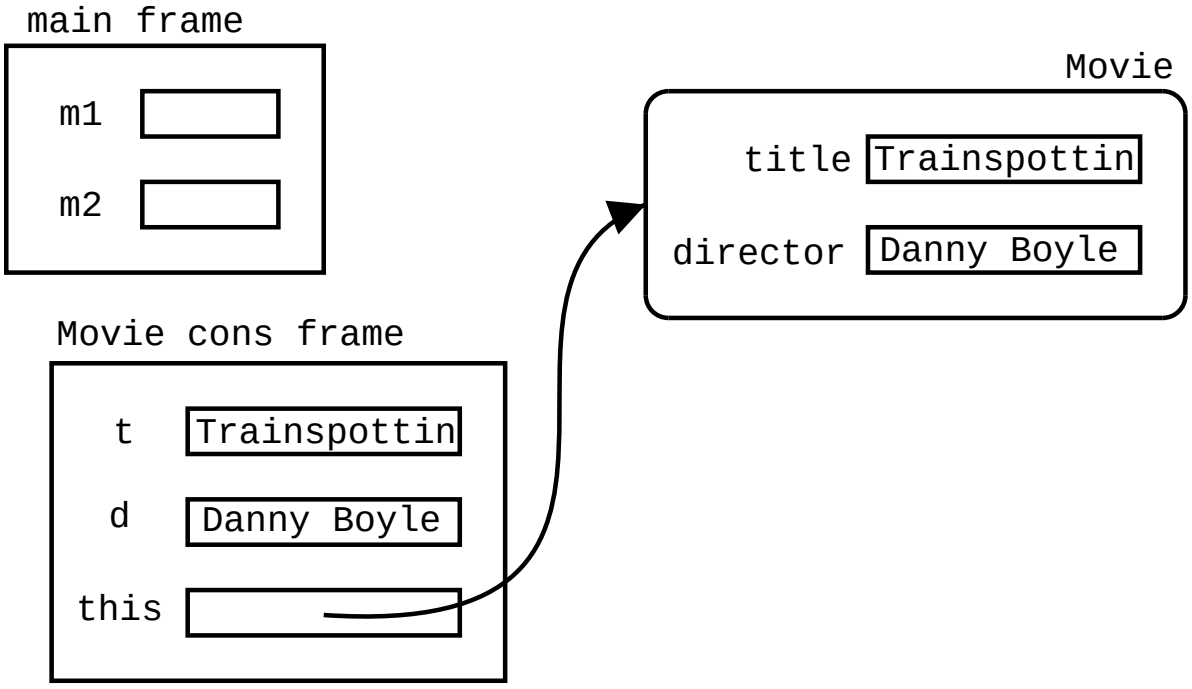
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

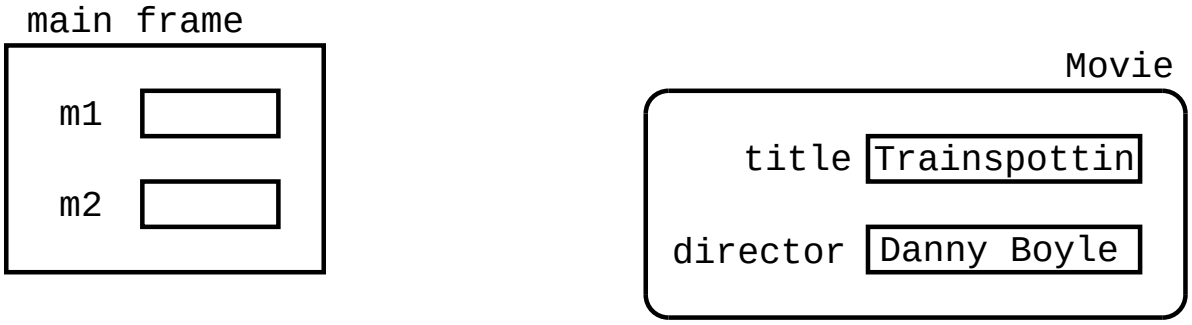
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

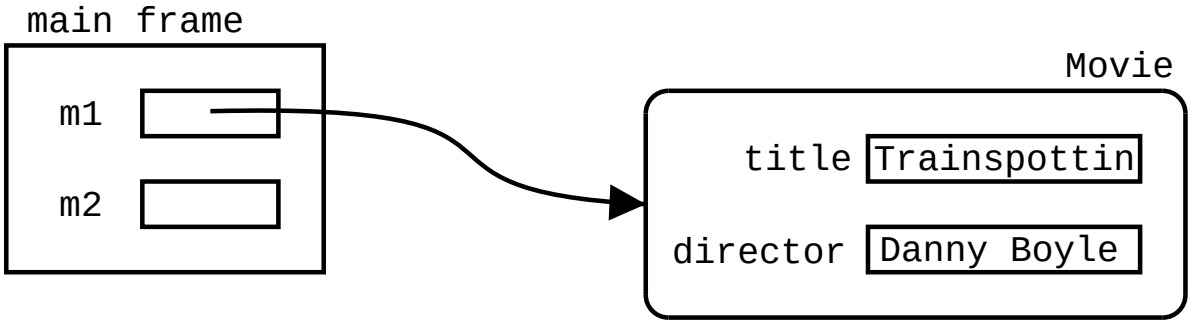
Method invocation: parameter passing



Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

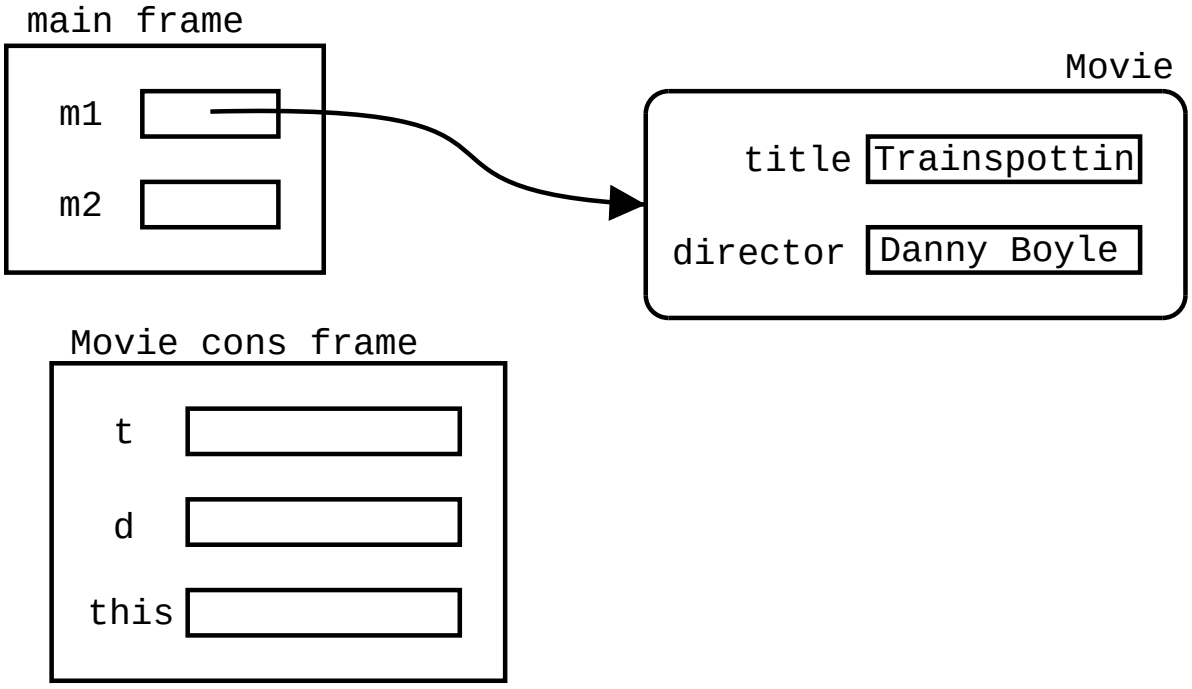
Method invocation: parameter passing



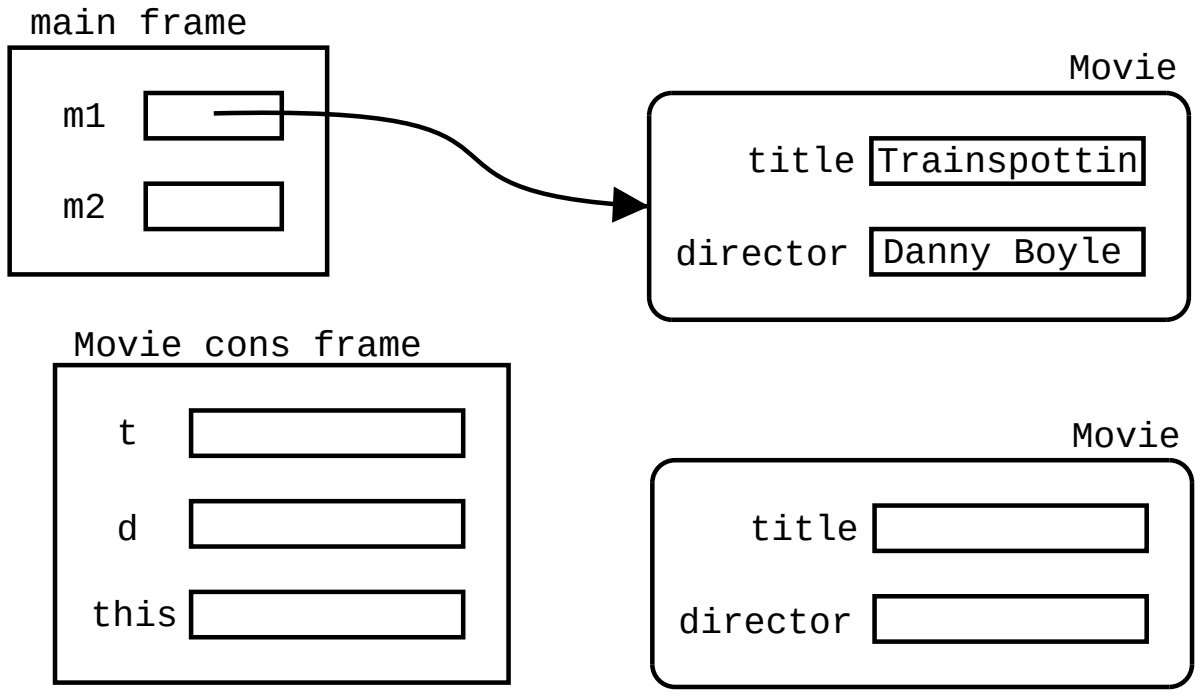
Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

Method invocation: parameter passing



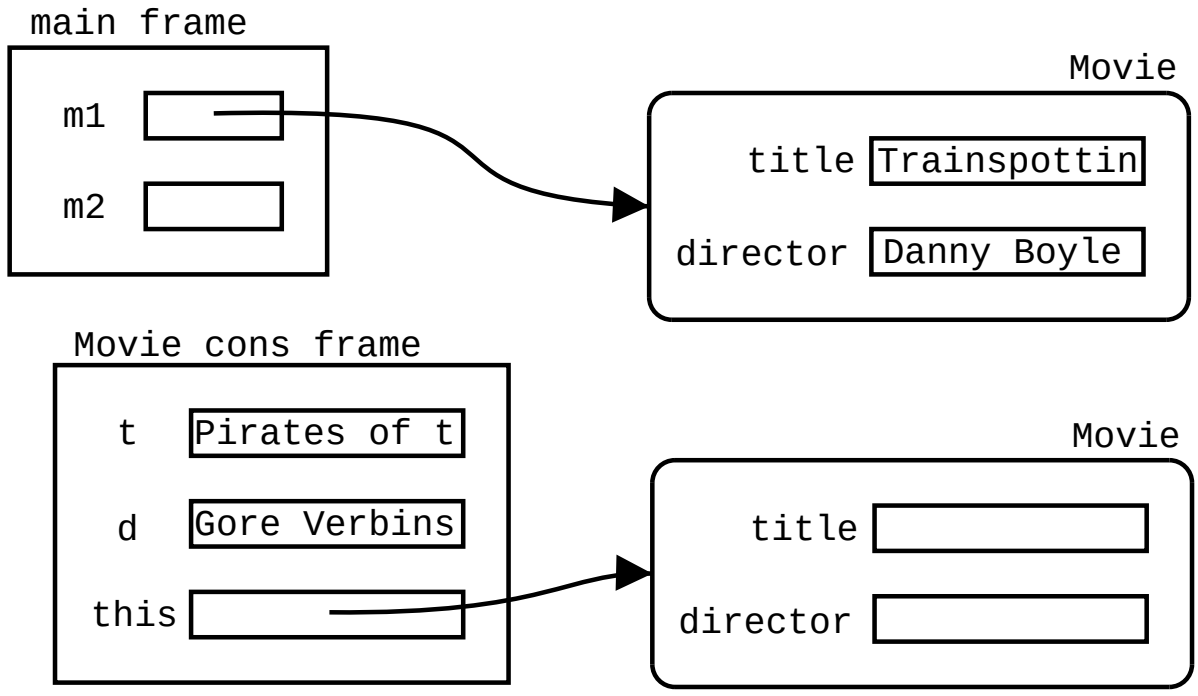
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

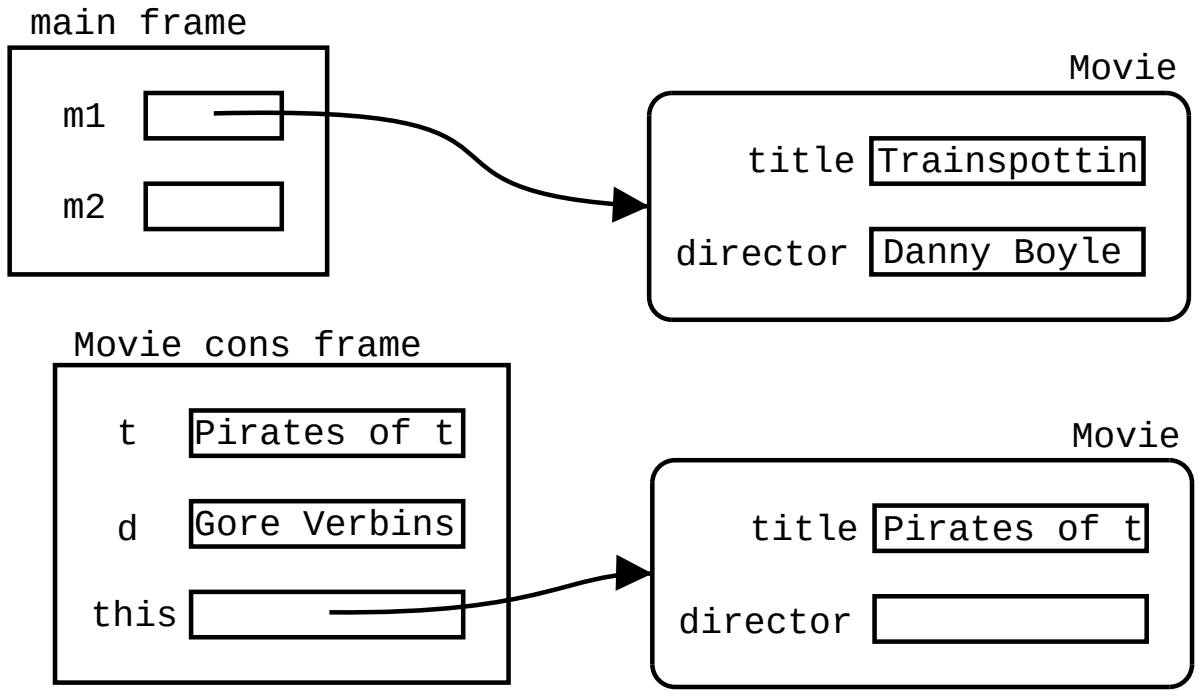
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

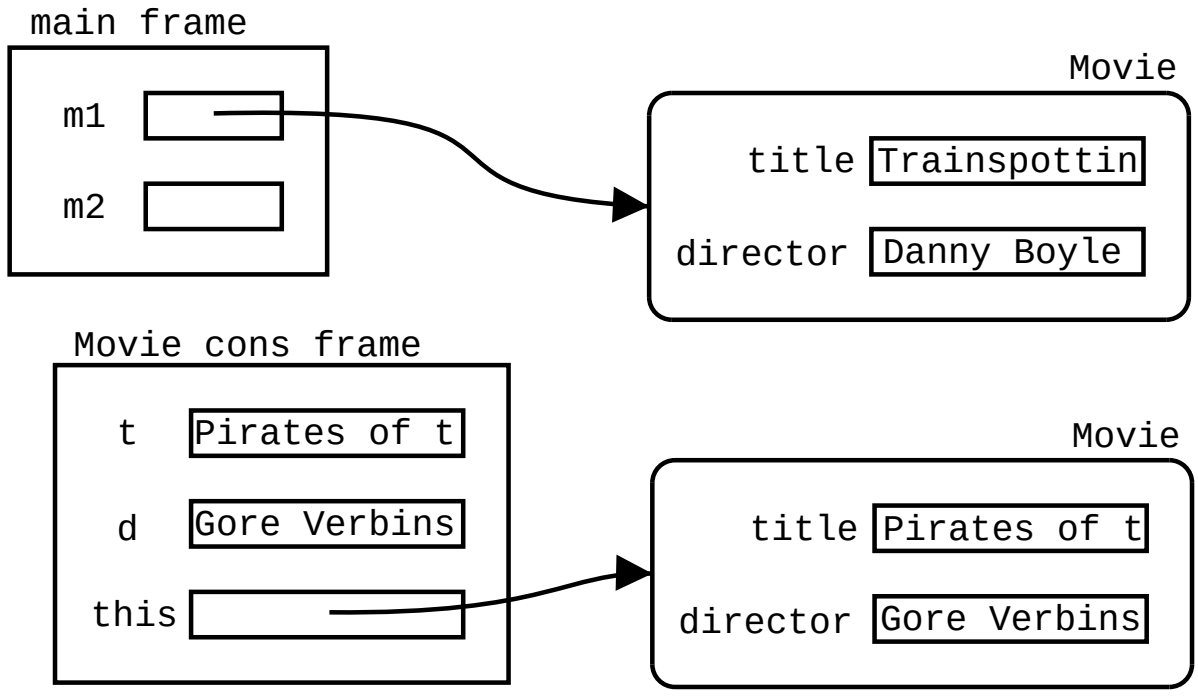
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

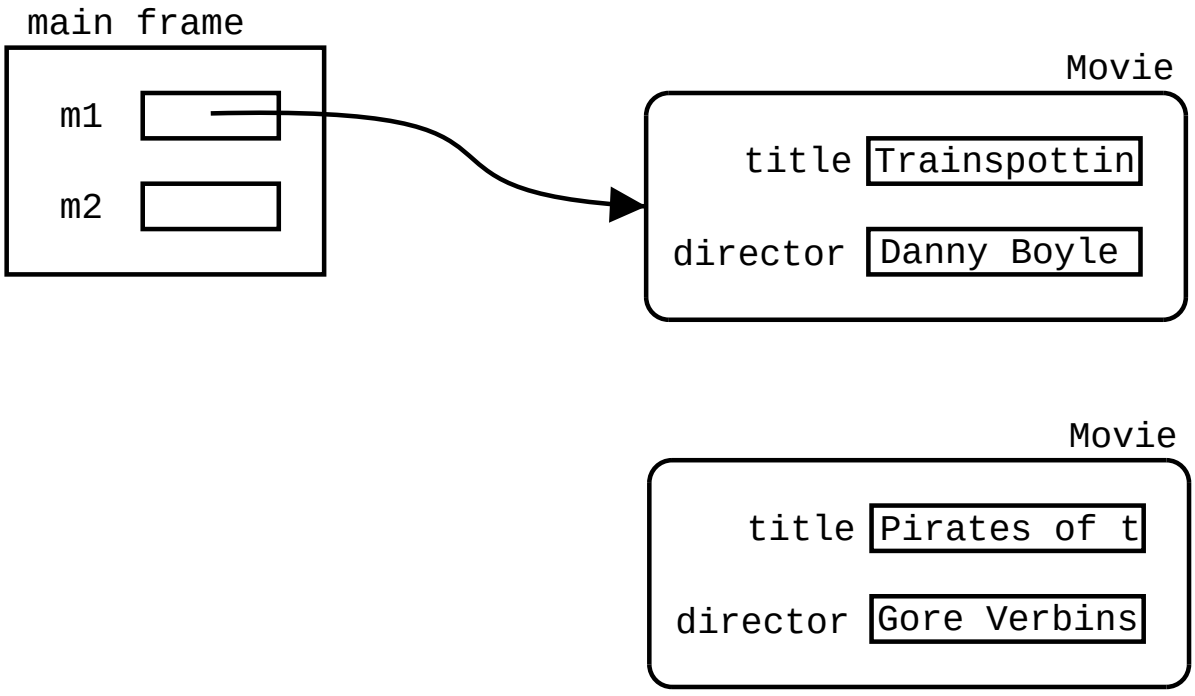
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

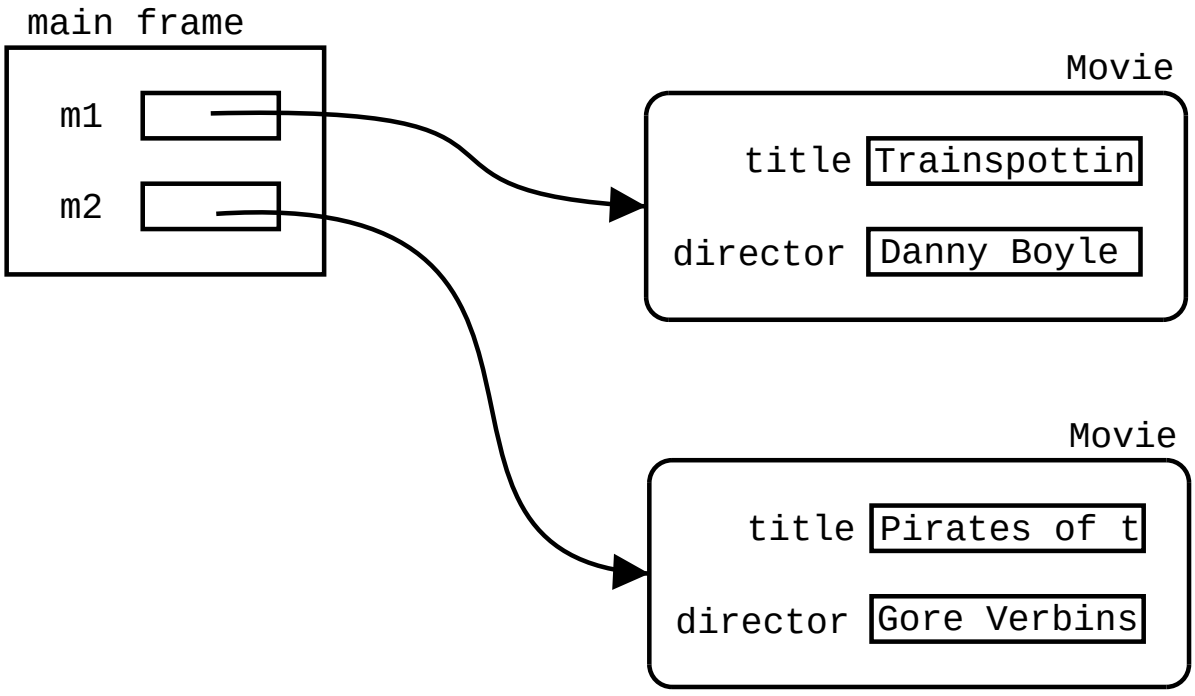
Method invocation: parameter passing



Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

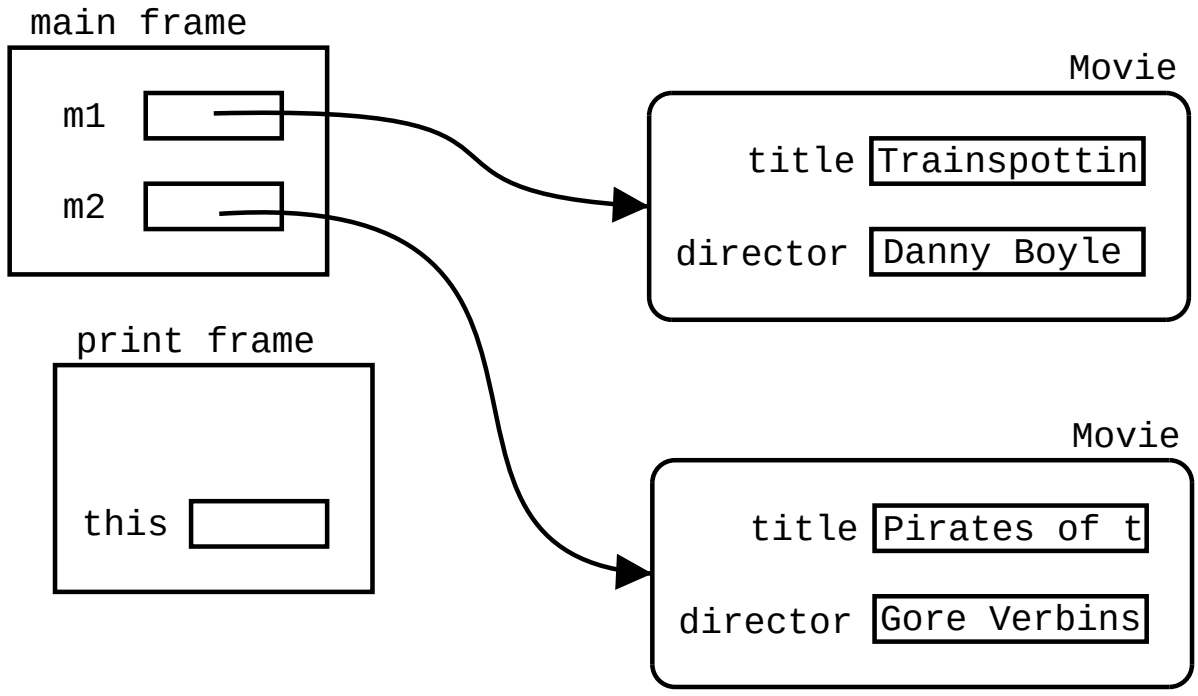
Method invocation: parameter passing



Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

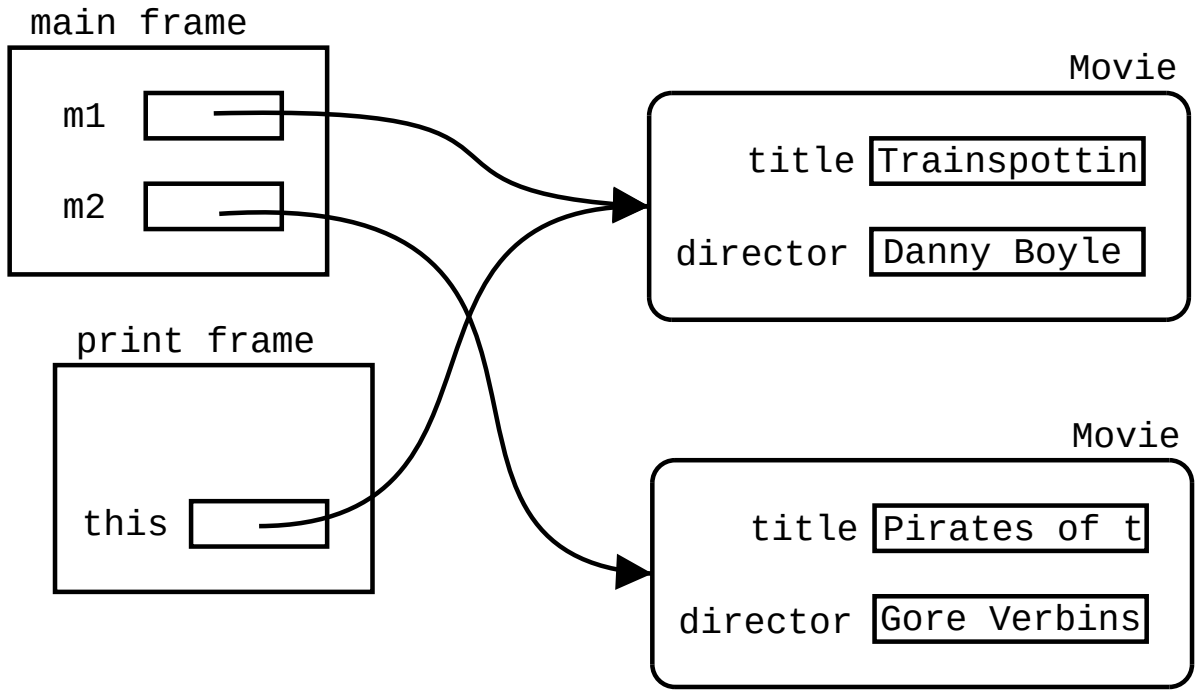
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

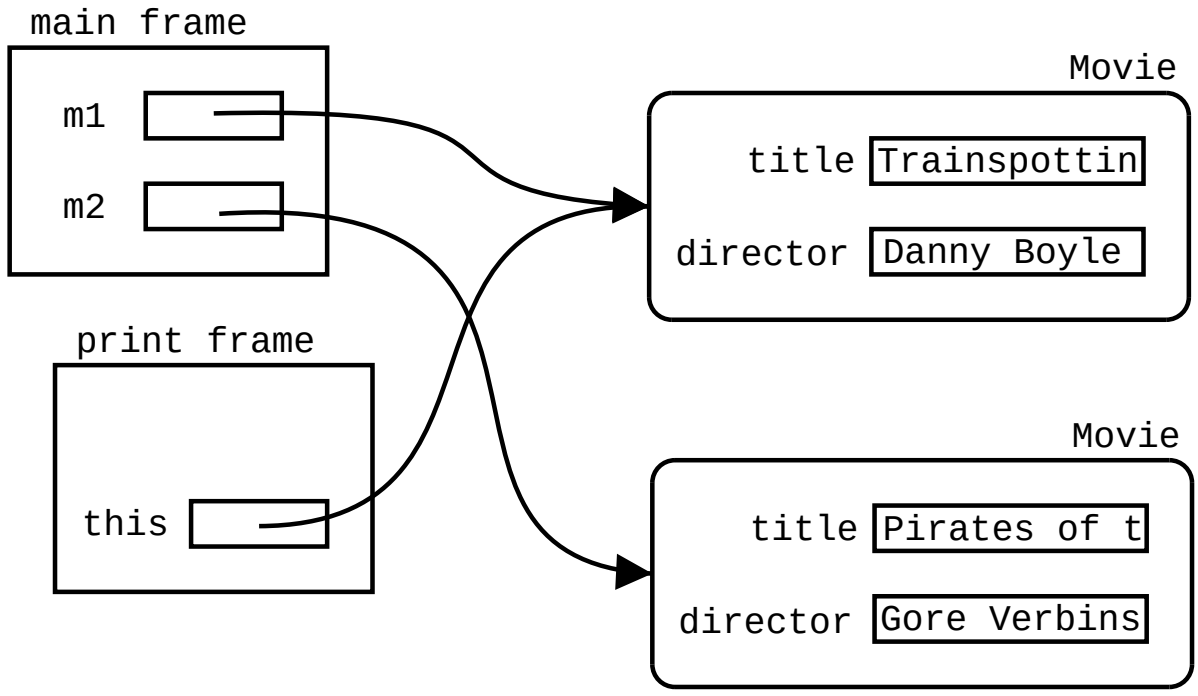
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

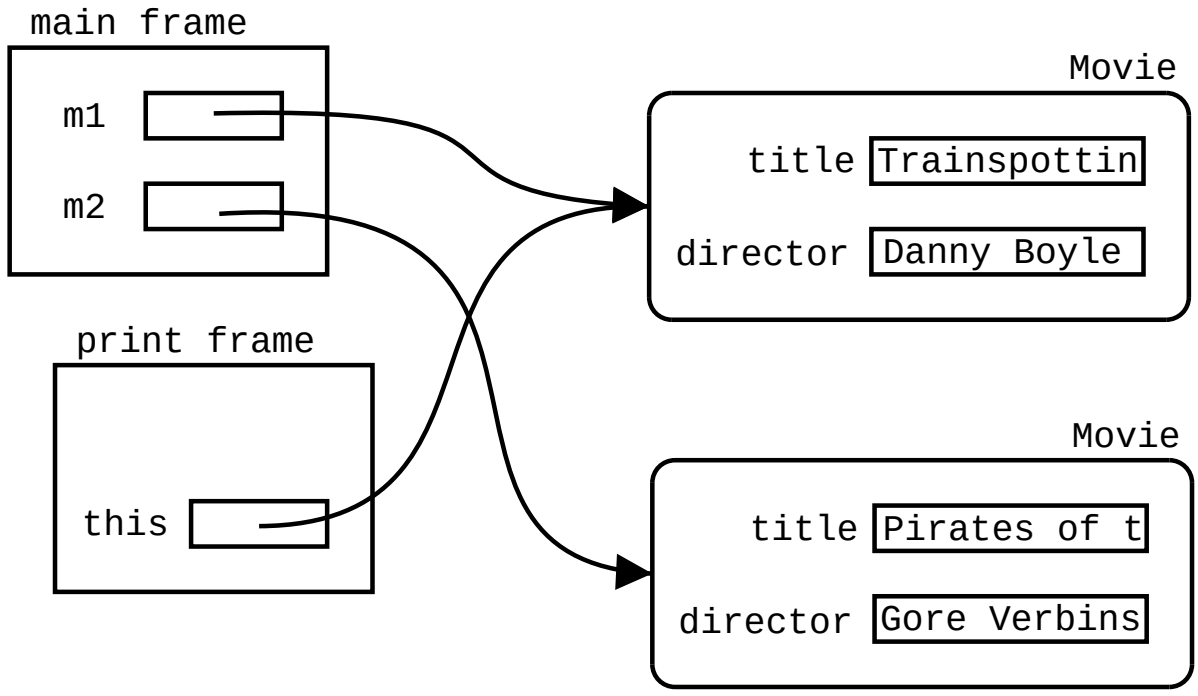
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

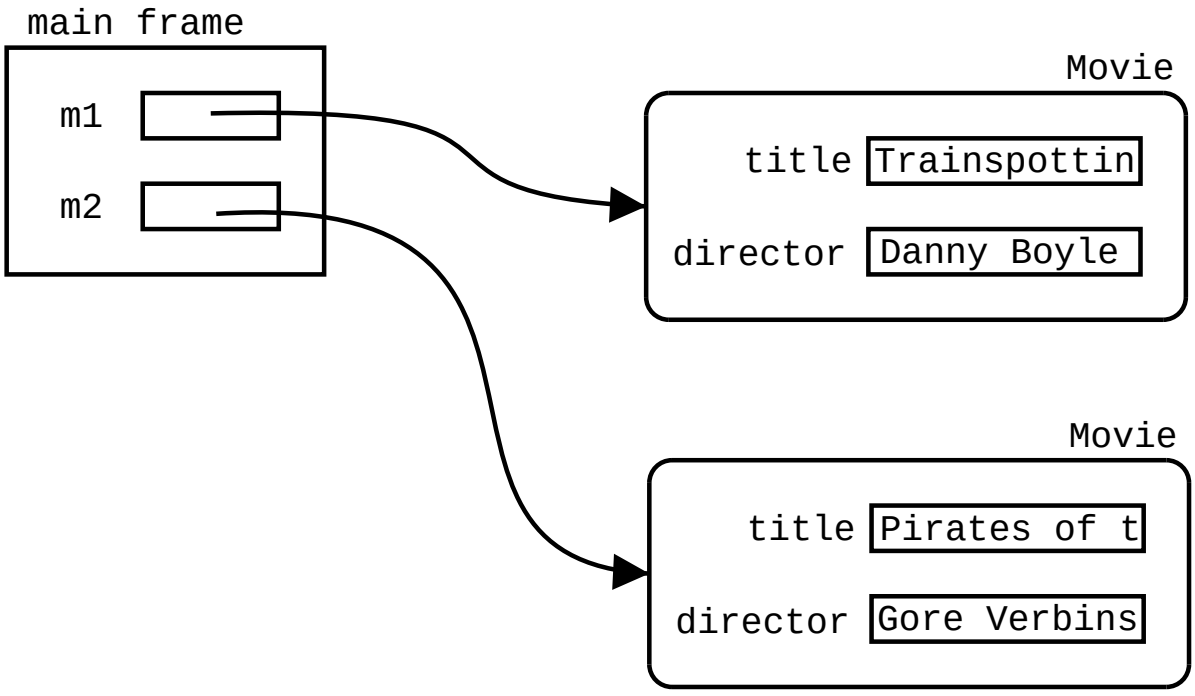
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

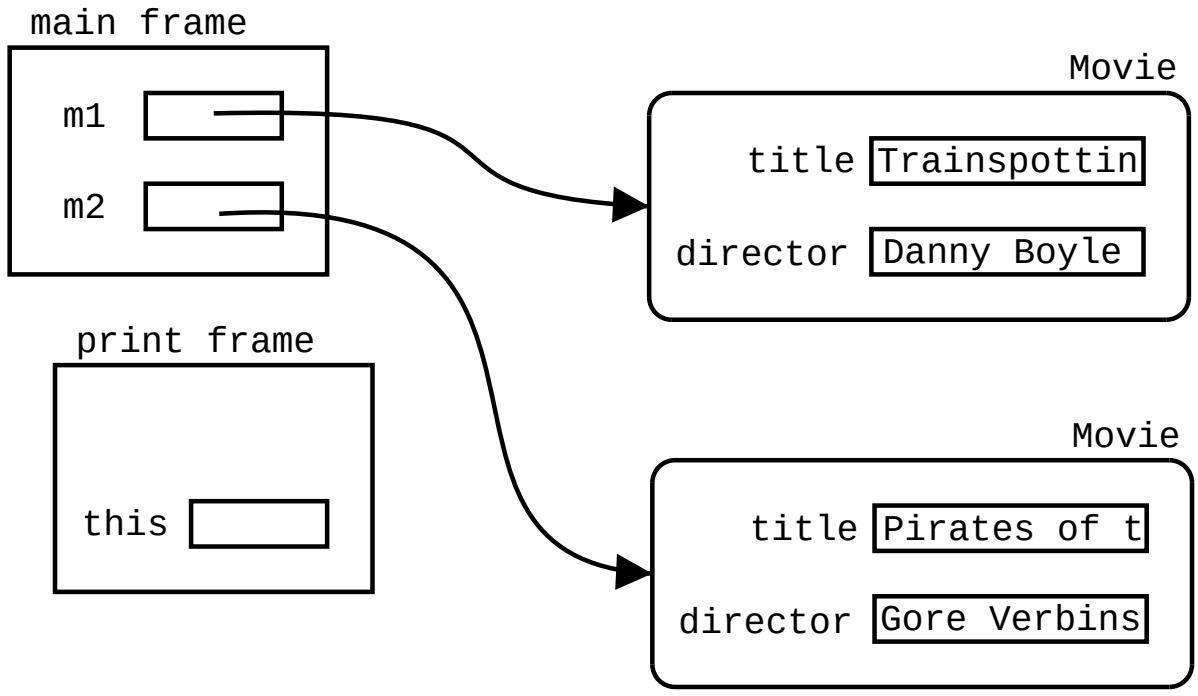
Method invocation: parameter passing



Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

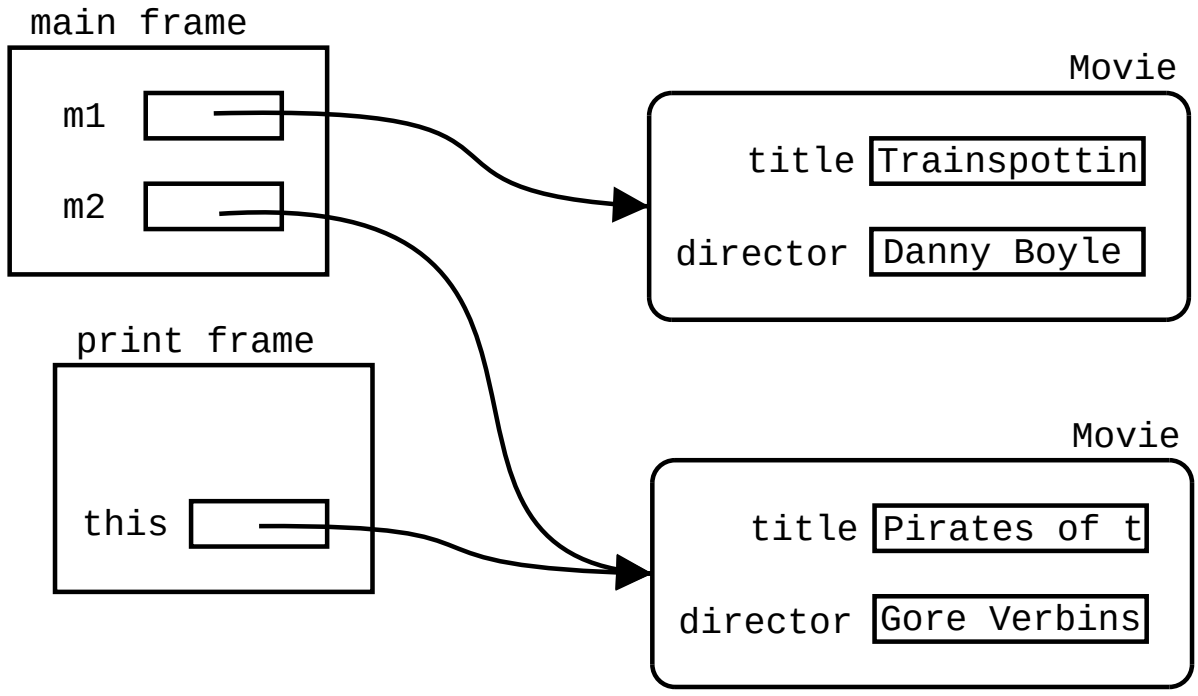
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

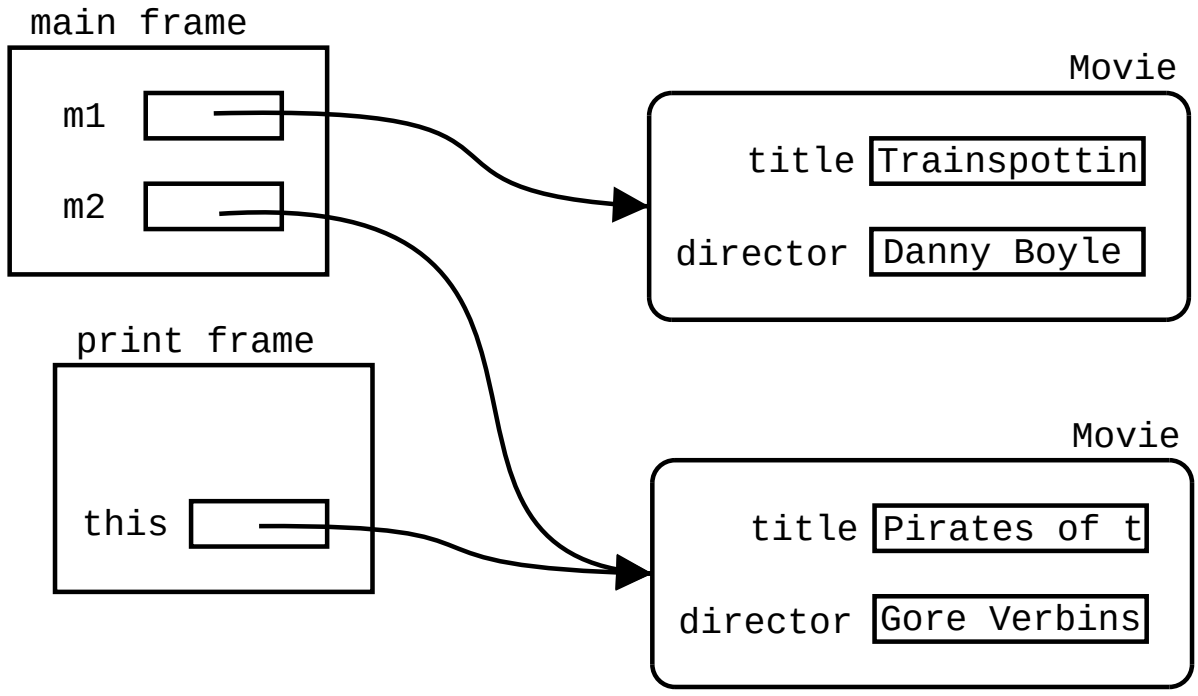
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

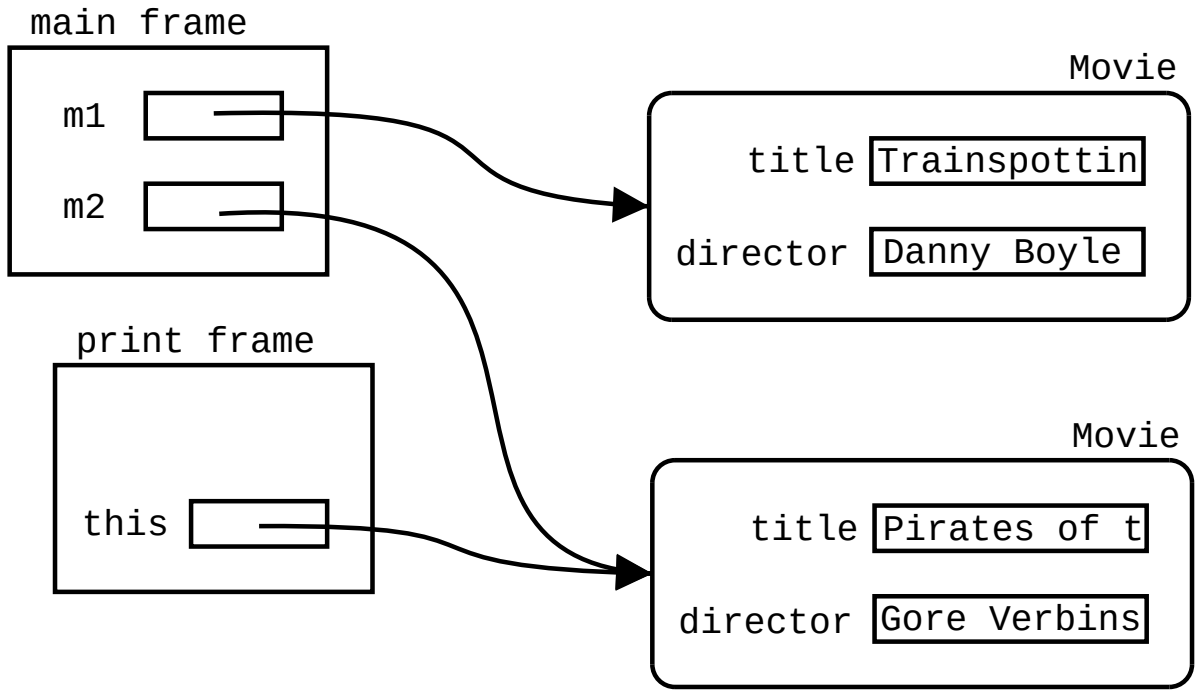
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

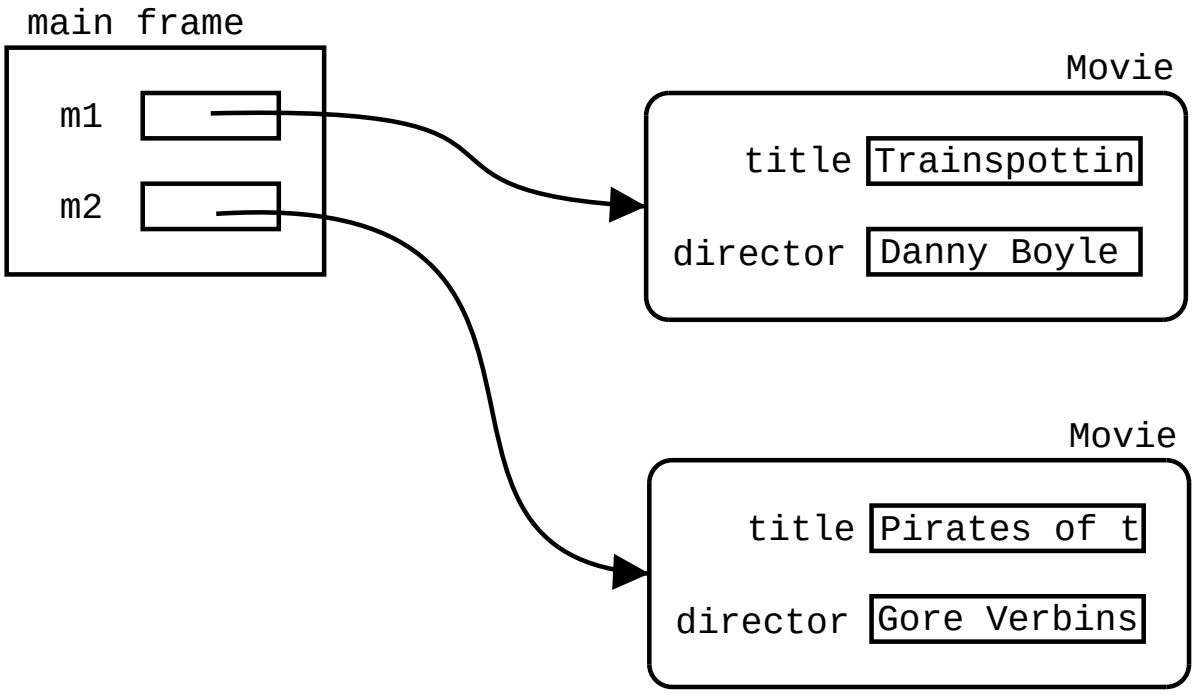
Method invocation: parameter passing



Method invocation: parameter passing

```
public class Movie
{
  String title, director;
  Movie(String t, String d)
  {
    this.title = t;
    this.director = d;
  }
  void print()
  {
    System.out.println(this.title);
    System.out.println(this.director);
  }
}
```

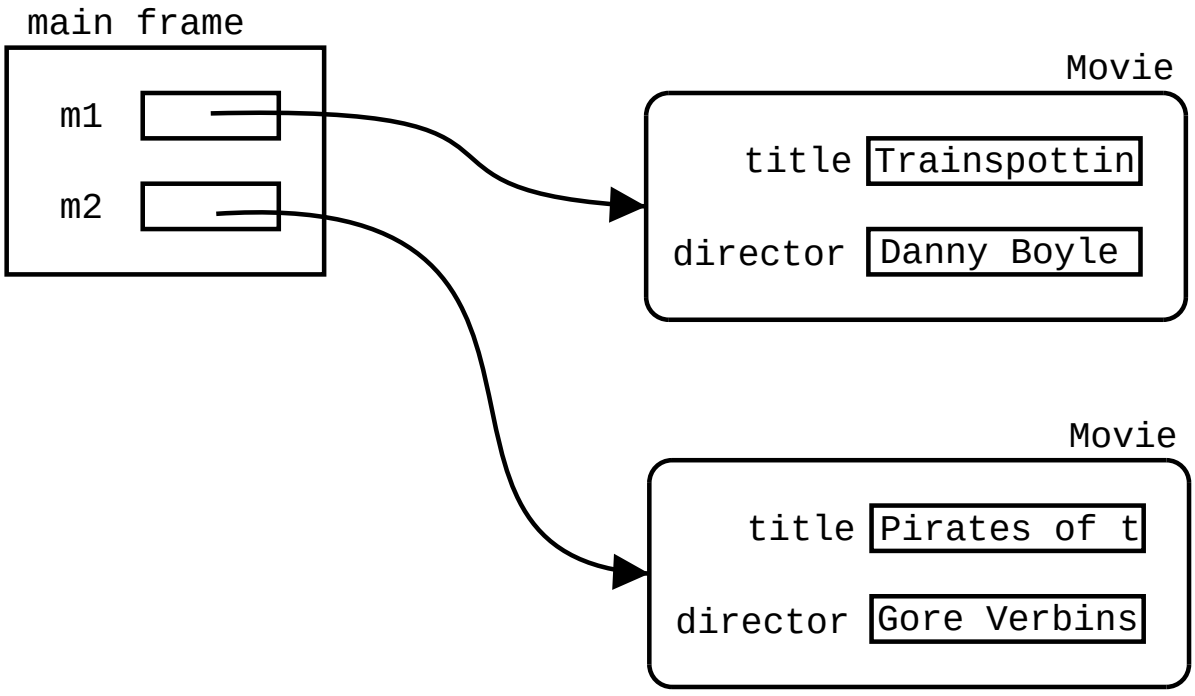
Method invocation: parameter passing



Method invocation: parameter passing

```
public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1, m2;
    m1 = new Movie(“Trainspotting”, “Danny Boyle”)
    m2 = new Movie(“Pirates of the Caribbean”,
                   “Gore Verbinski”);
    m1.print();
    m2.print();
  }
}
```

Method invocation: parameter passing



Method invocation

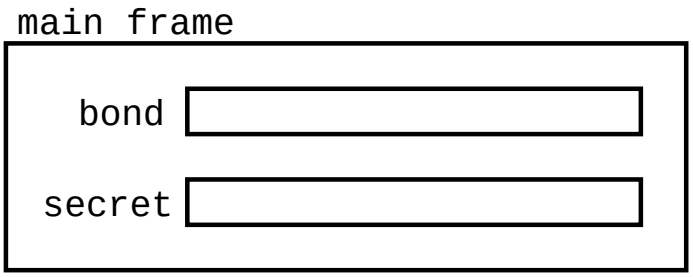
```
public class Spy
{
  int id;
  String name;

  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”);
  }
}
```

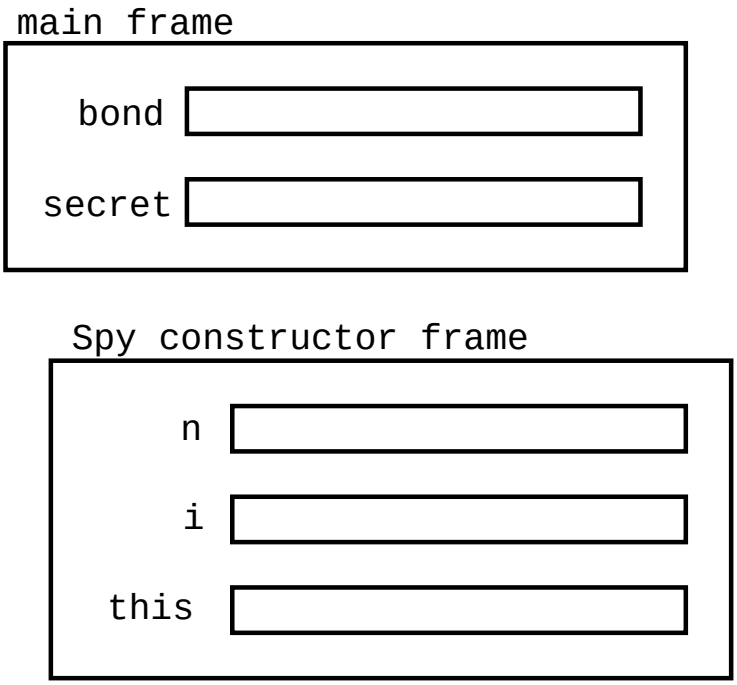
Method invocation



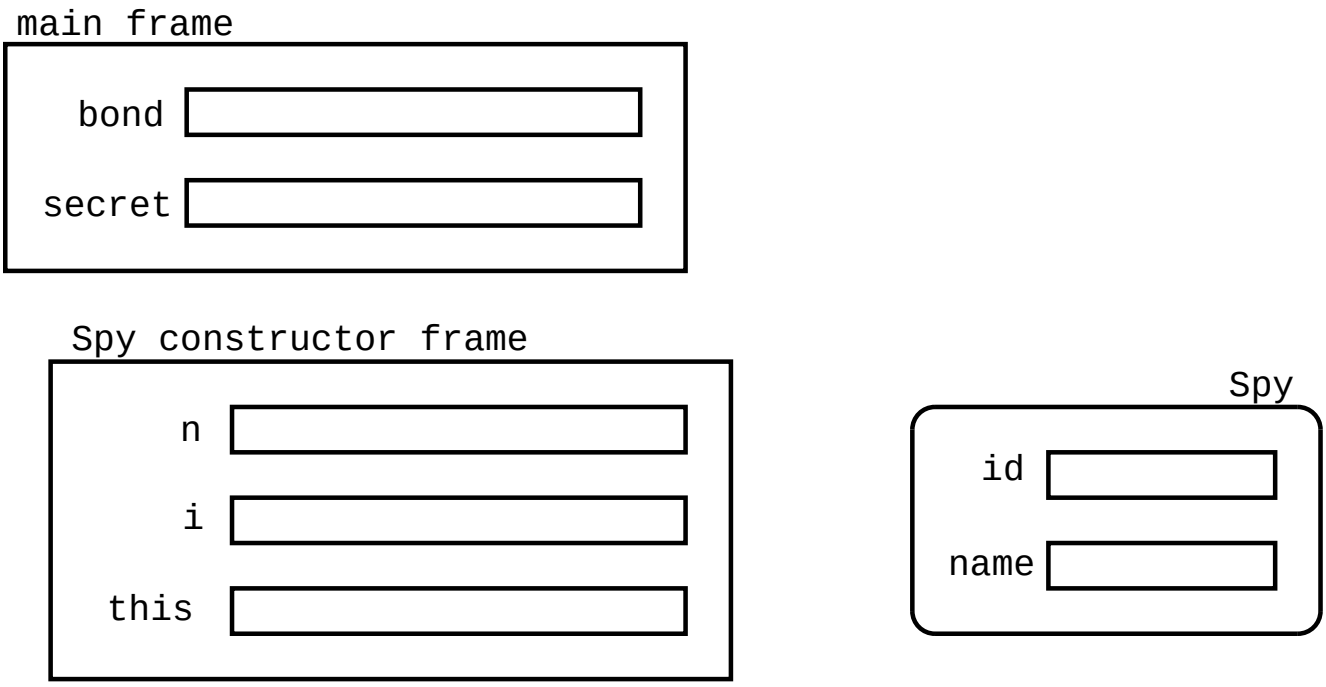
Method invocation

```
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



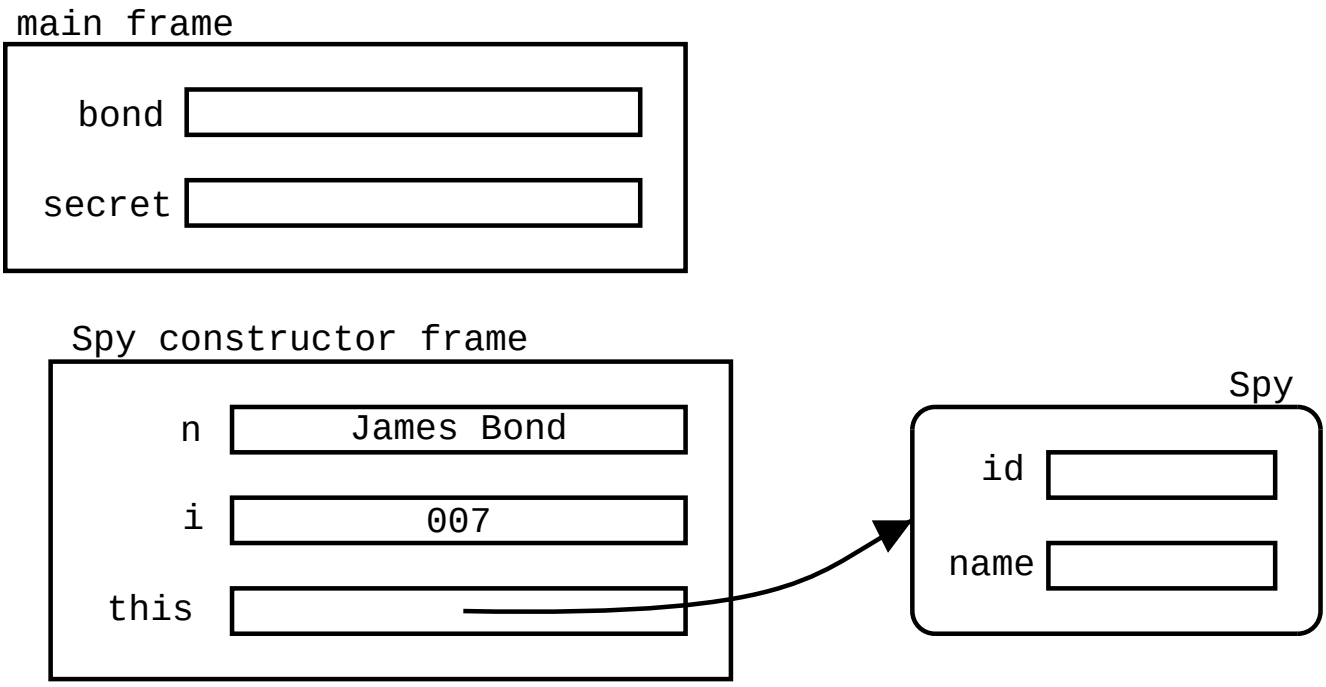
Method invocation



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;
  }
}
```

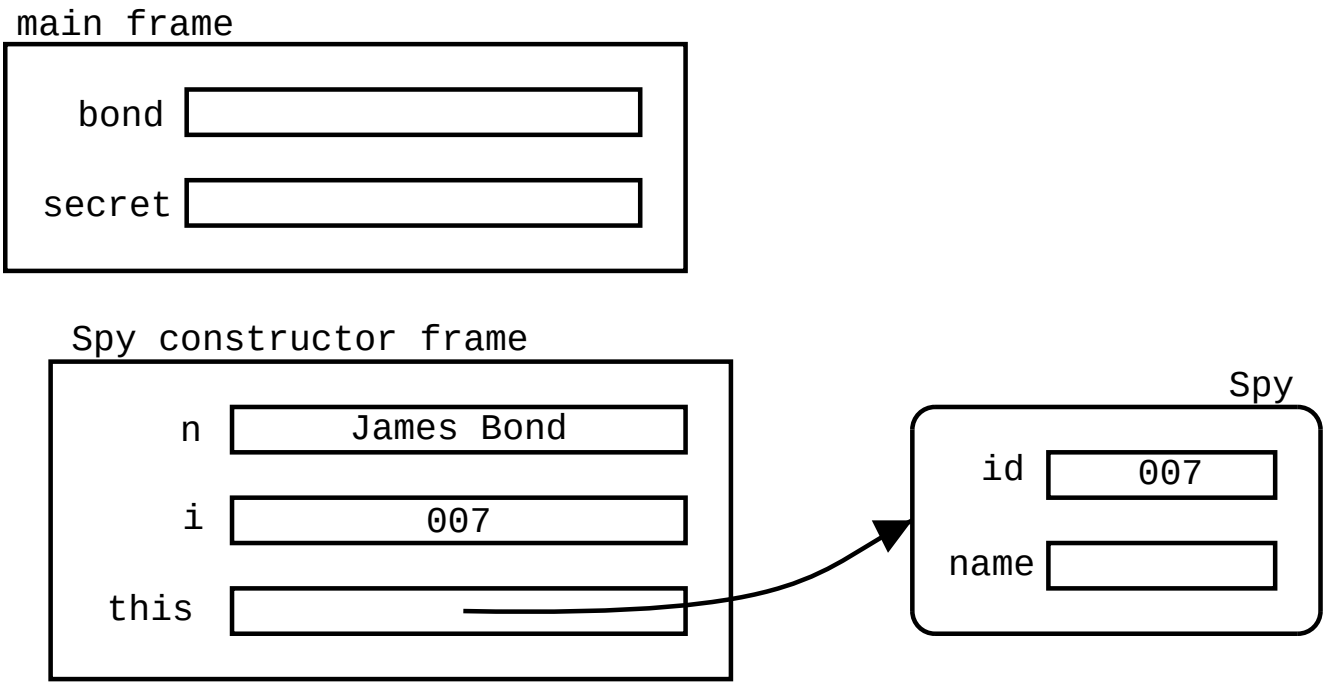
Method invocation



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;
  }
}
```

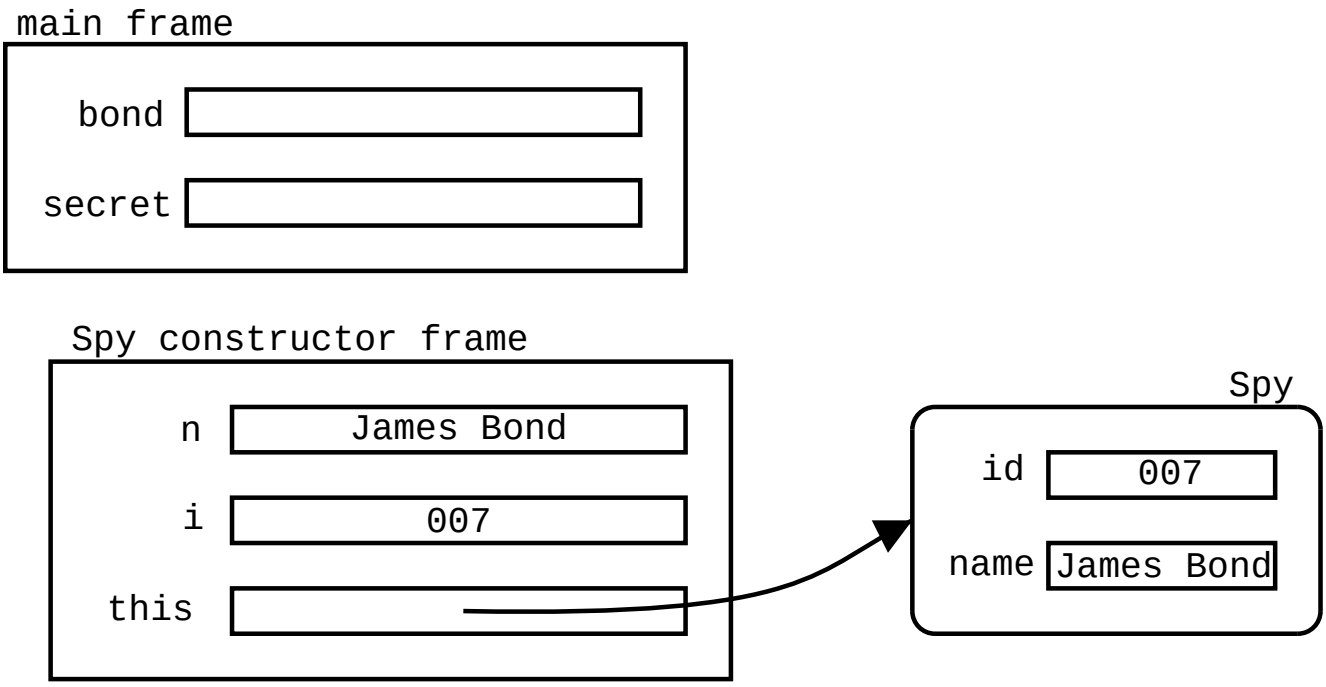
Method invocation



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



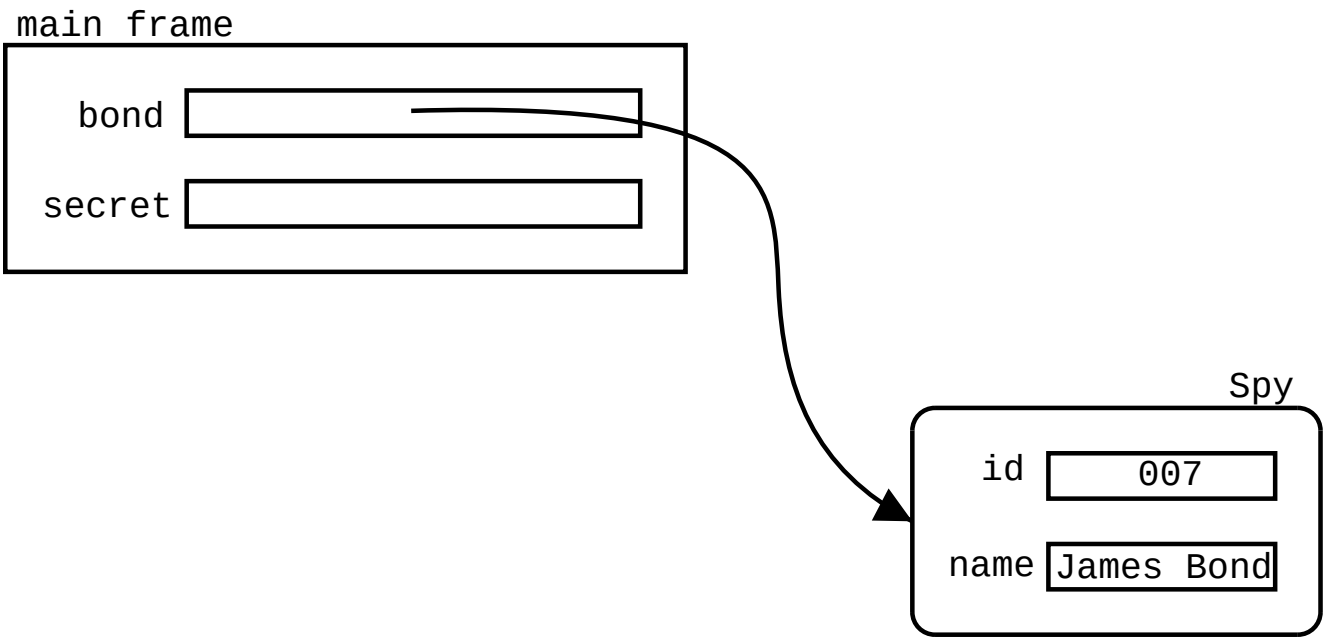
Method invocation

```
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”);
  }
}
```

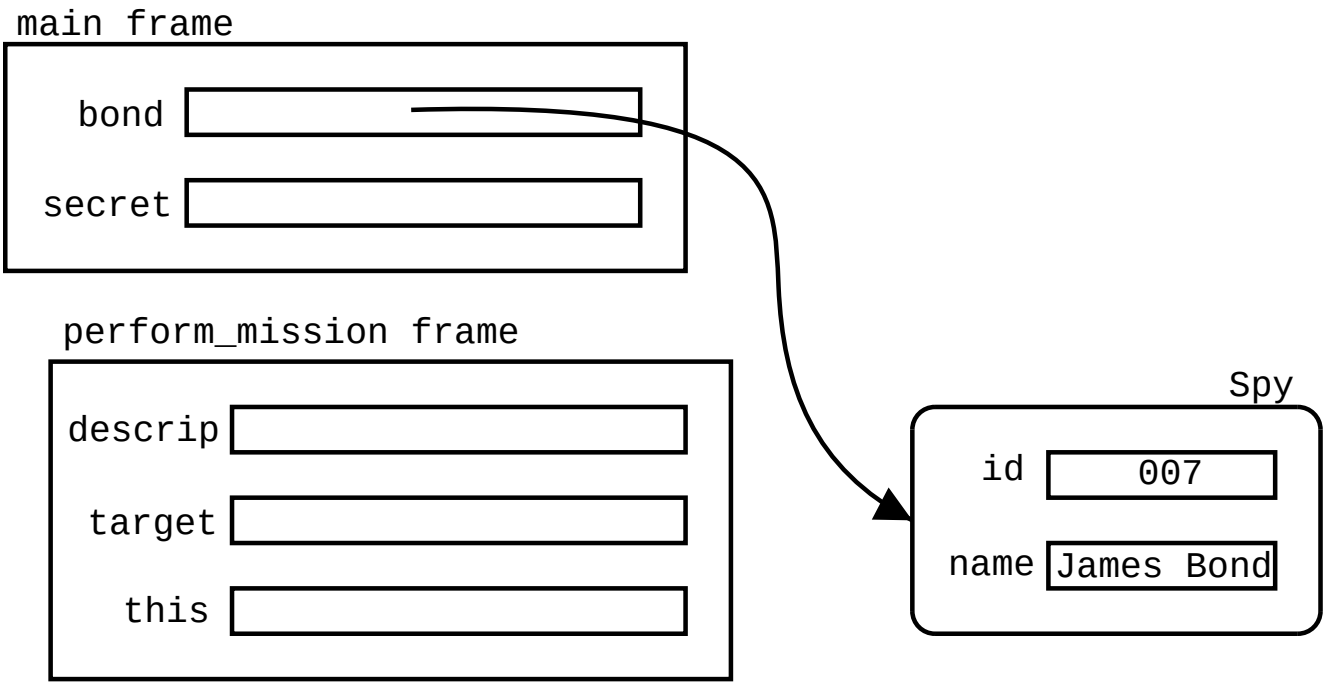
Method invocation



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”);
  }
}
```

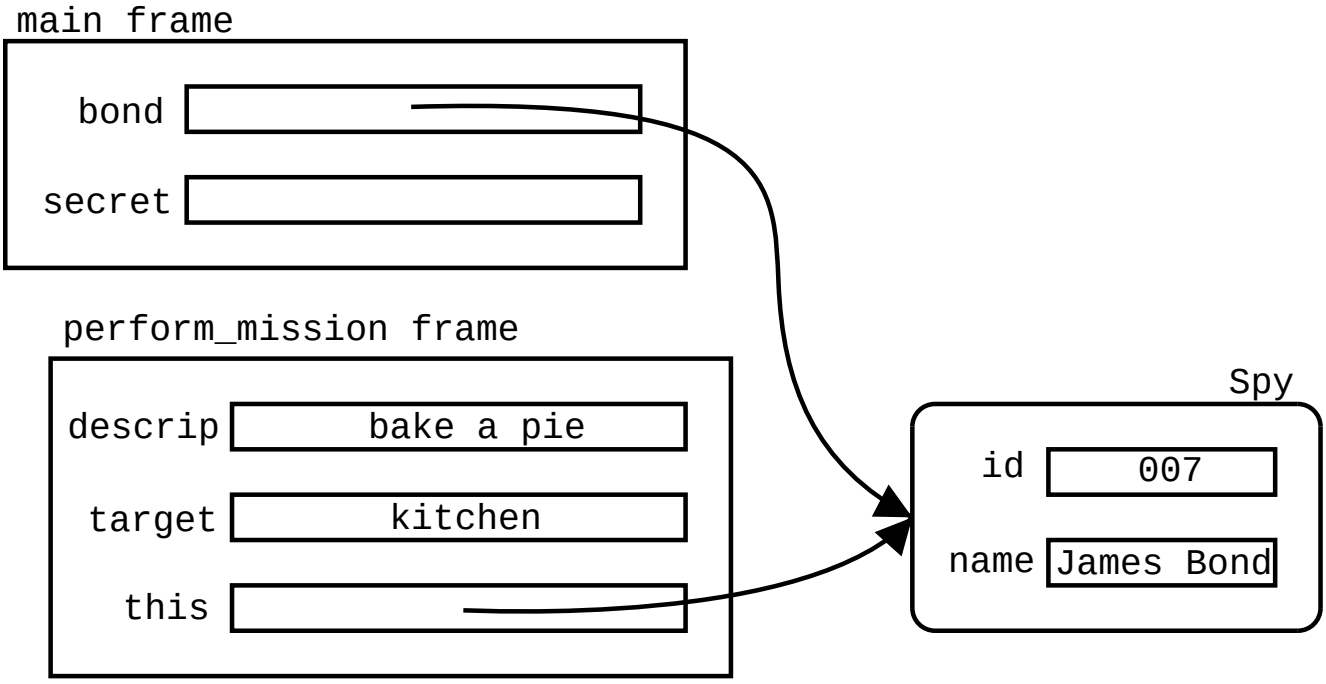
Method invocation



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;
  }
}
```

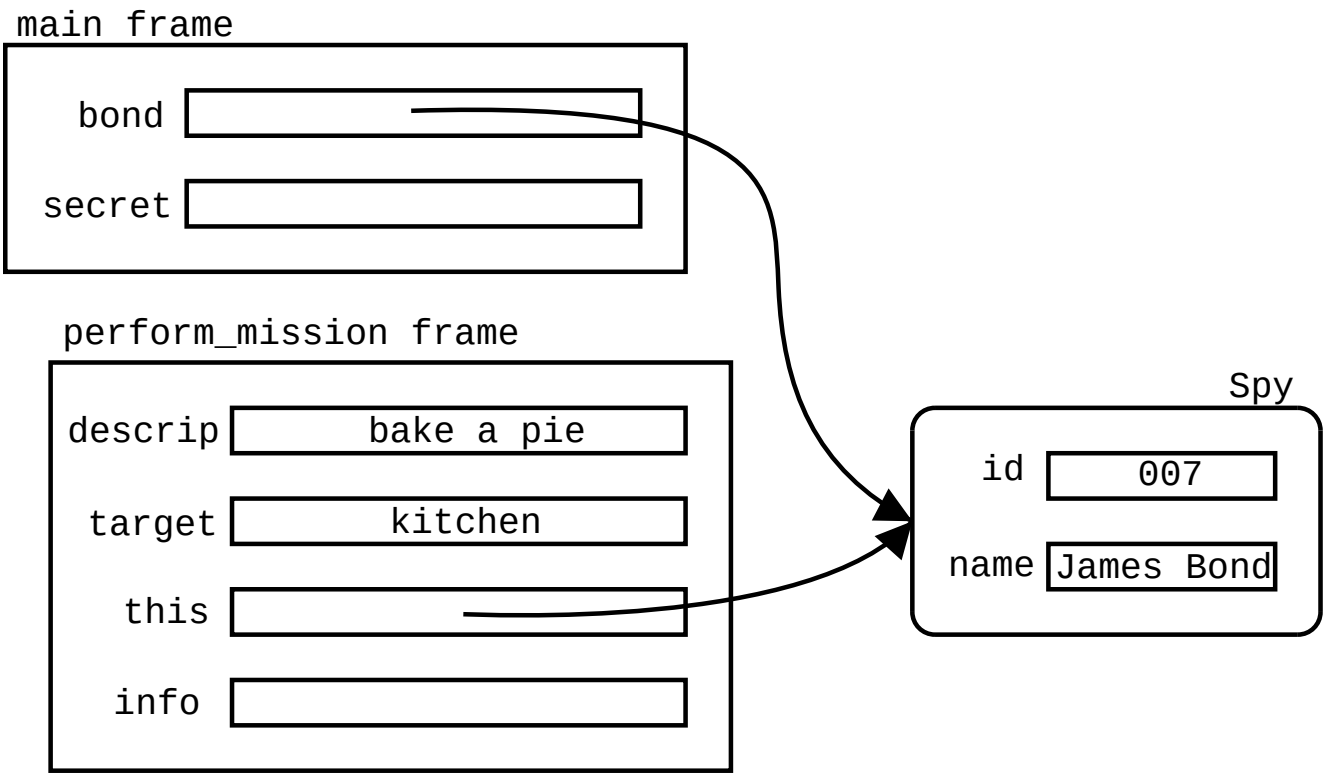
Method invocation



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;
  }
}
```

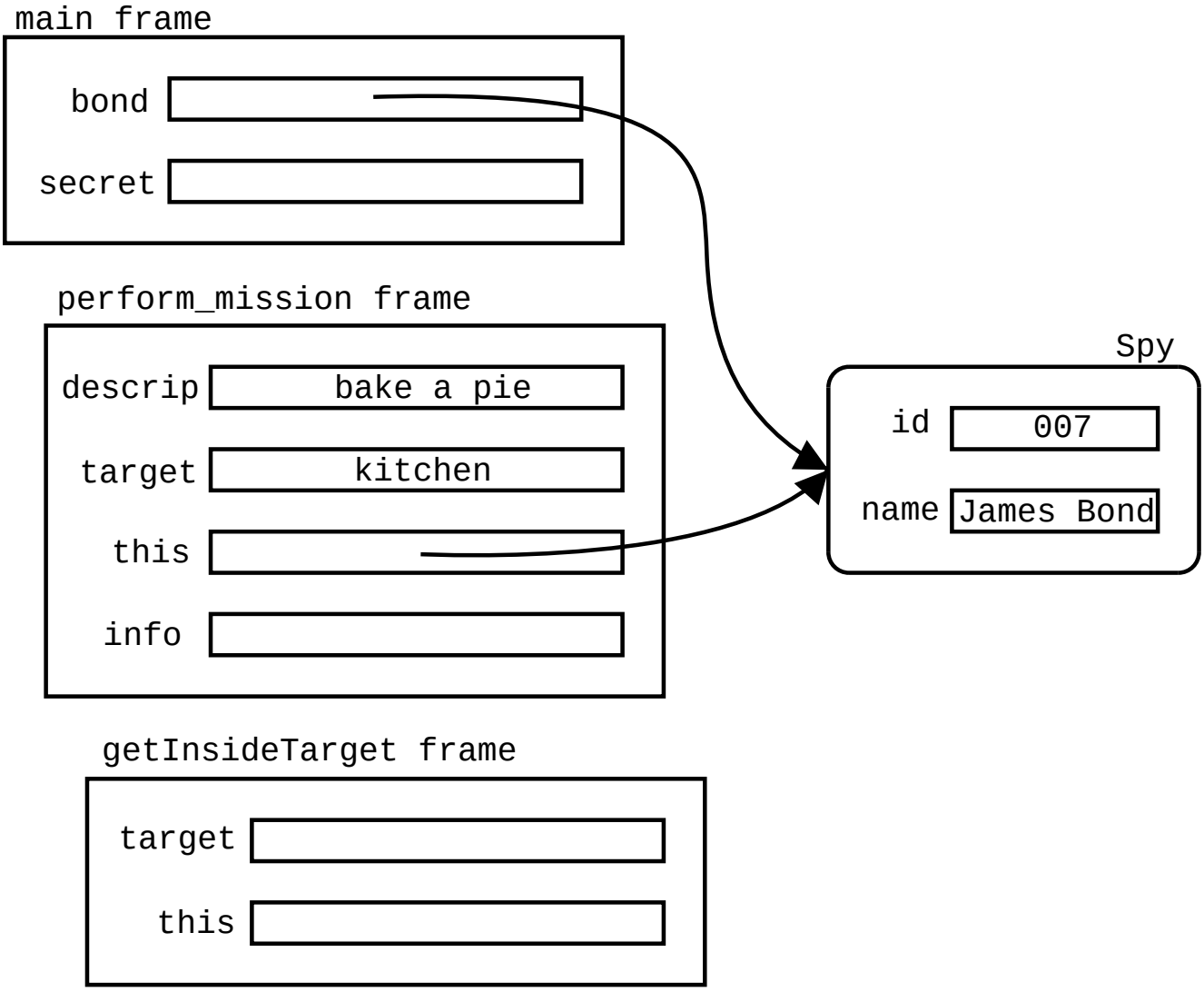
Method invocation



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;
  }
}
```

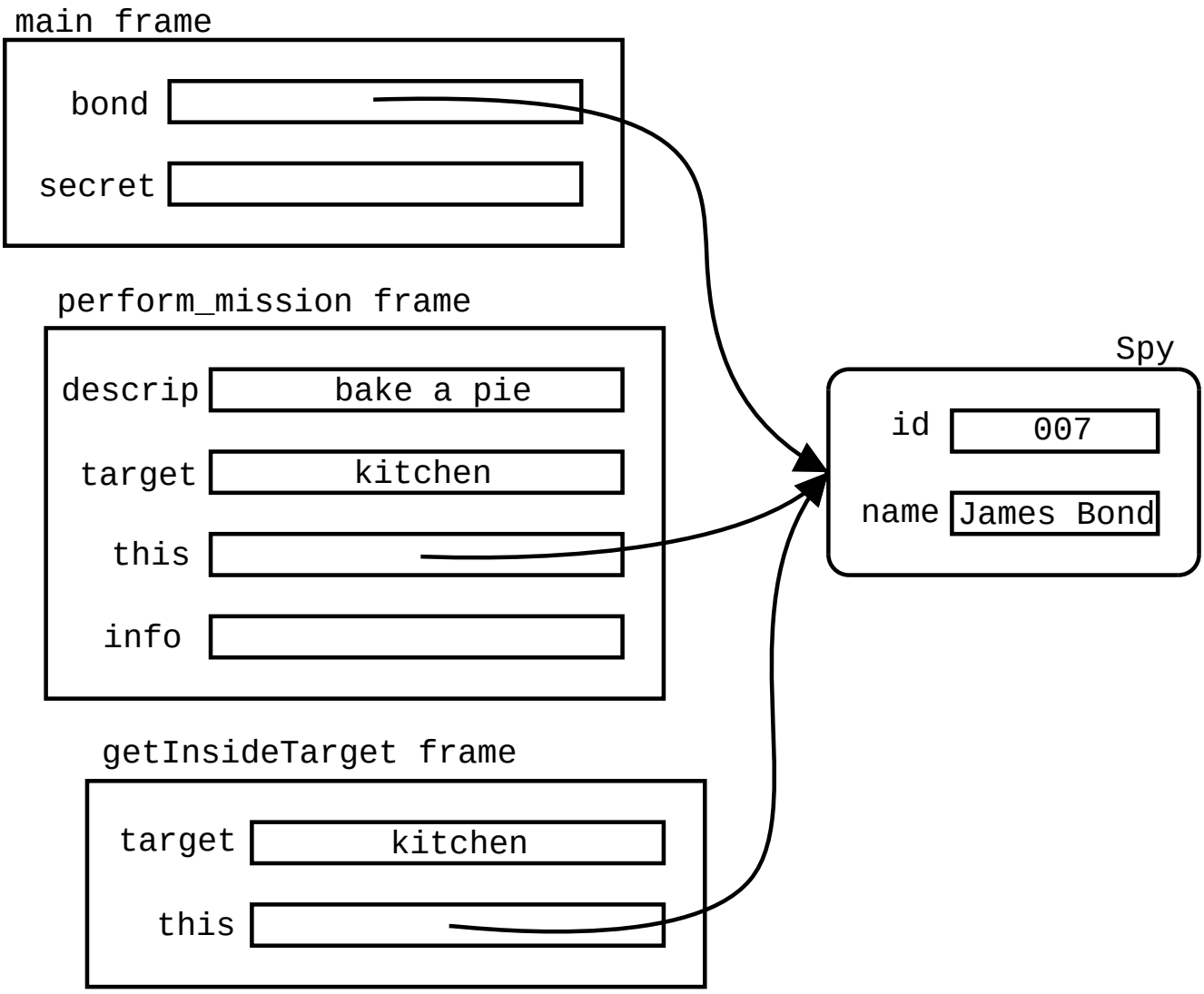
Method invocation



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;
  }
}
```

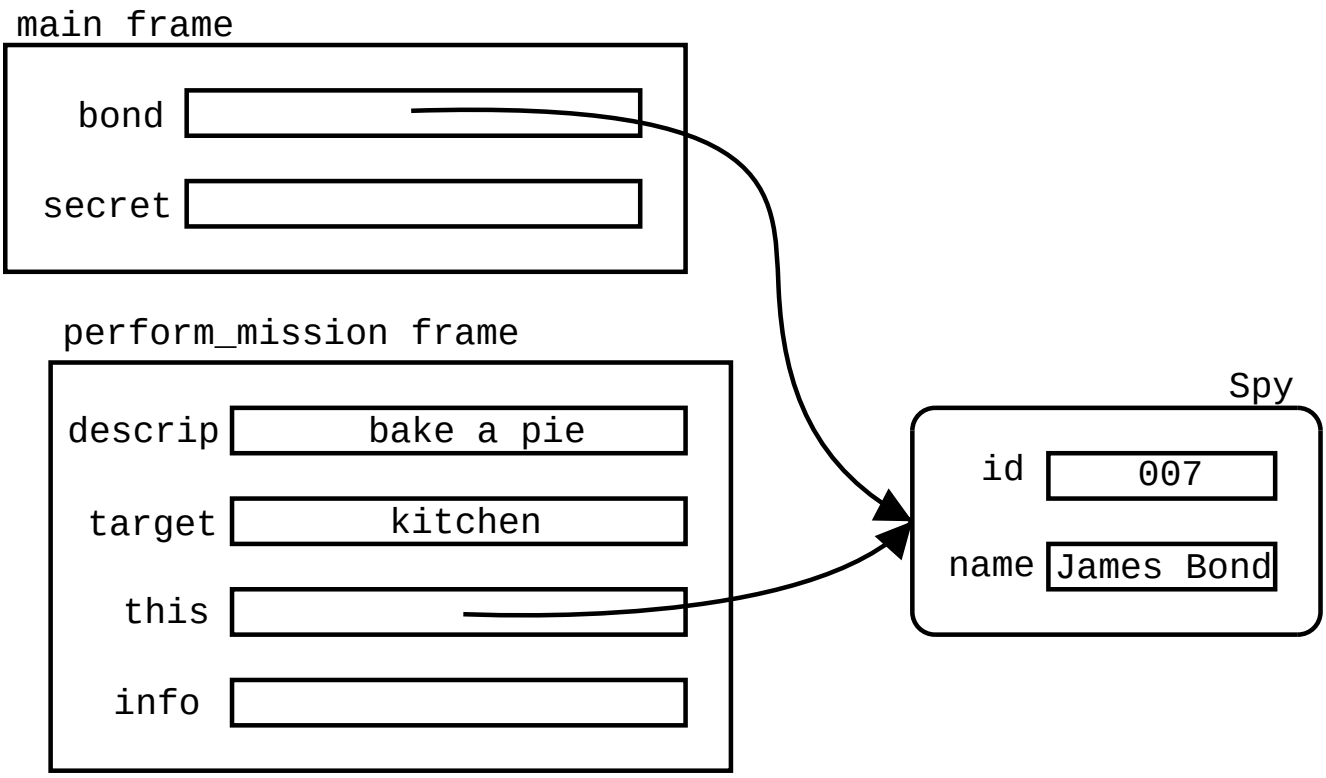
Method invocation

```
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;
  }
}
```

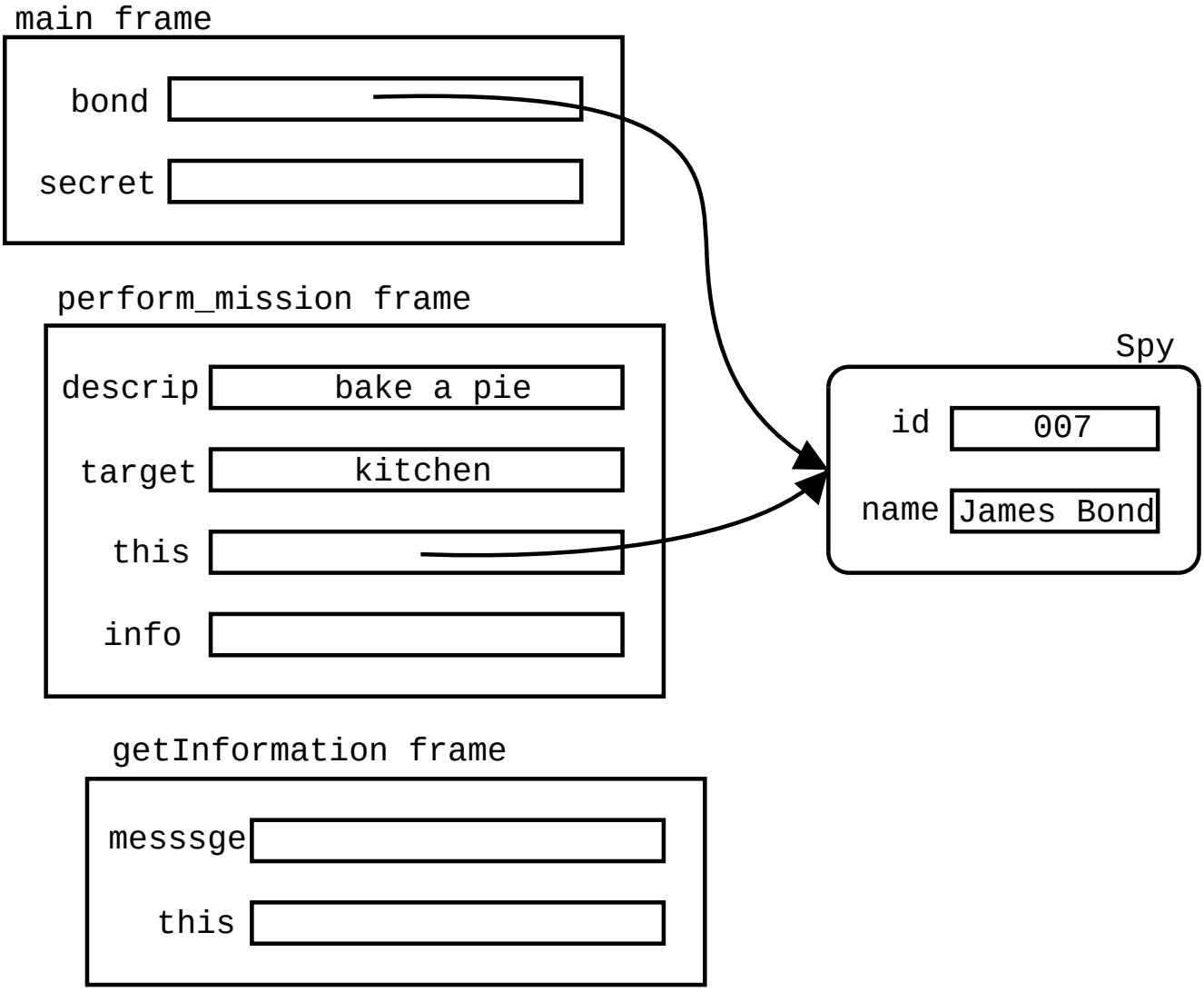
Method invocation



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;
  }
}
```

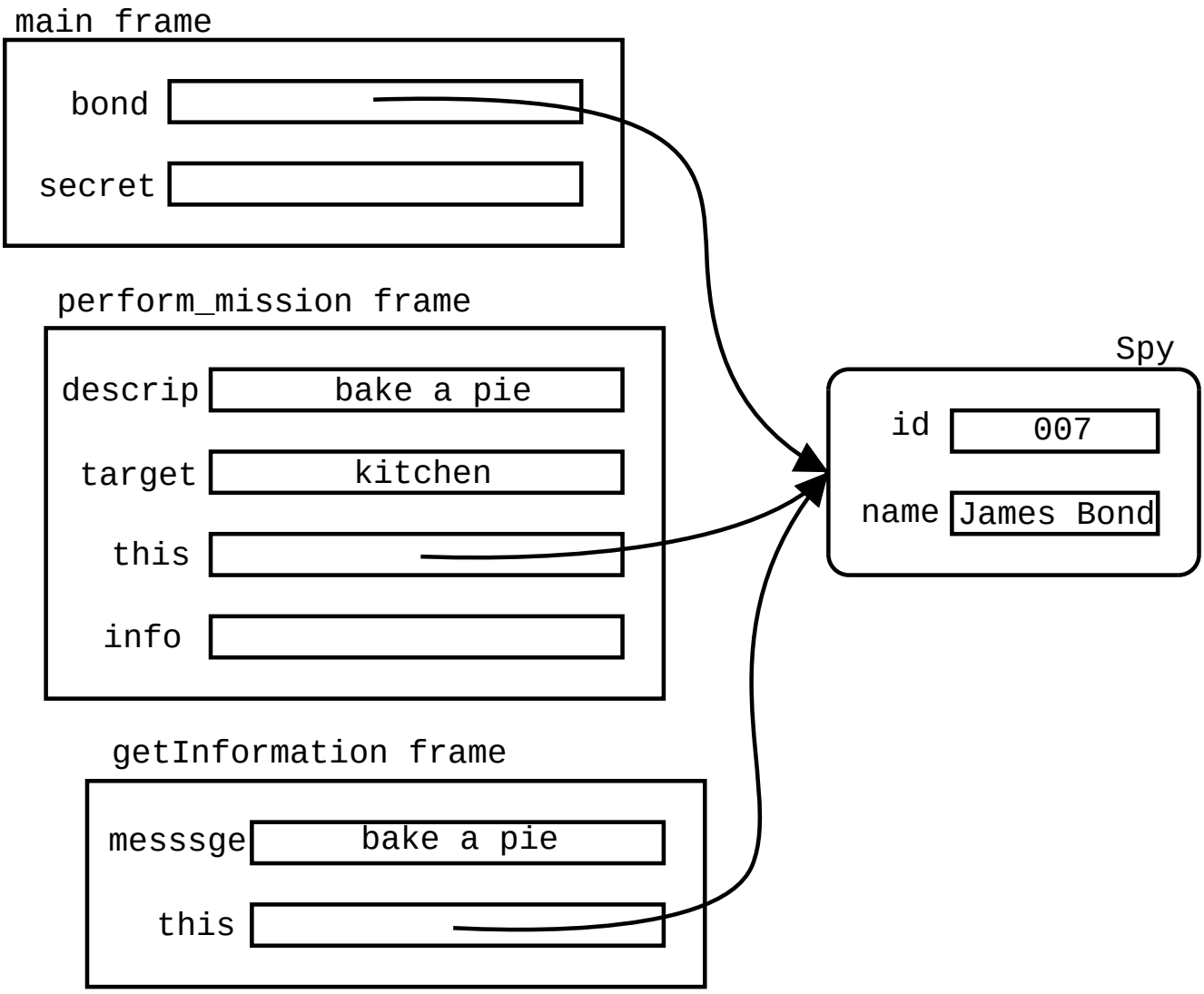
Method invocation



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;
  }
}
```

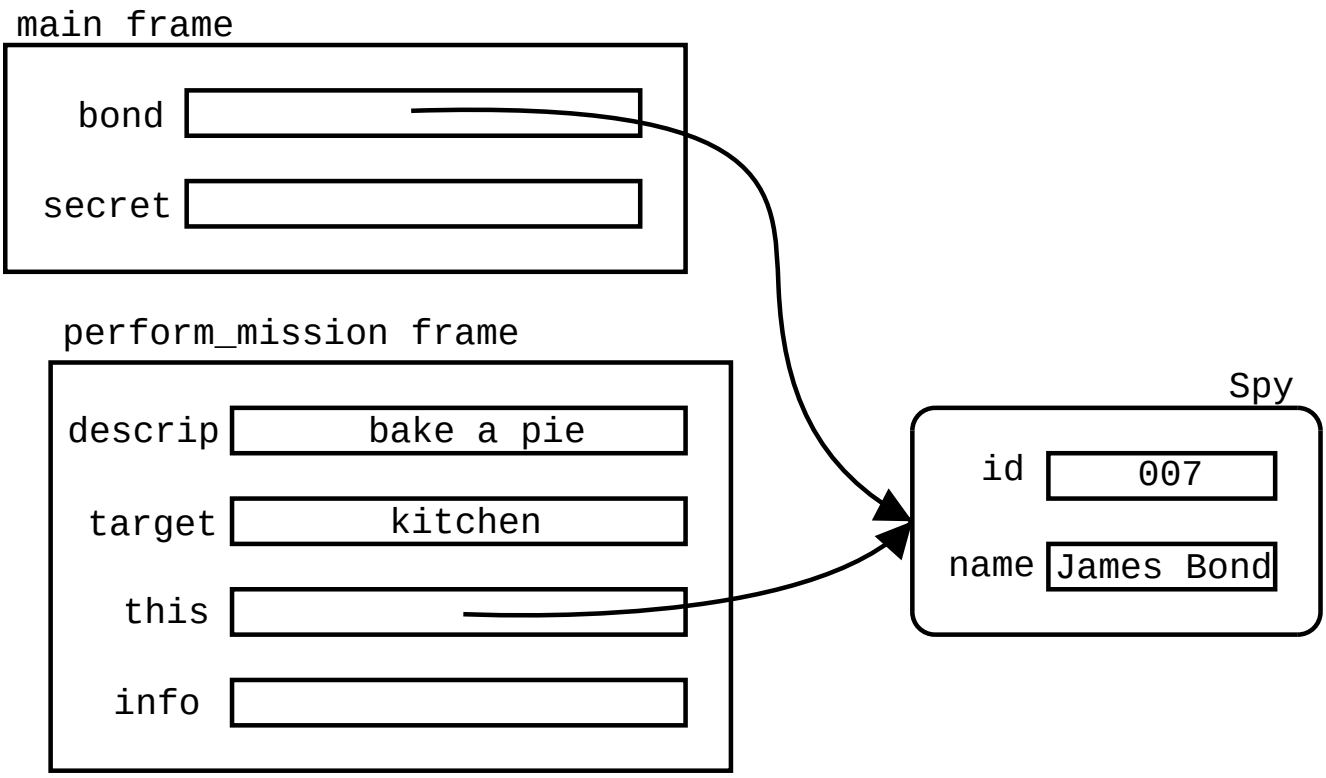
Method invocation



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;
  }
}
```

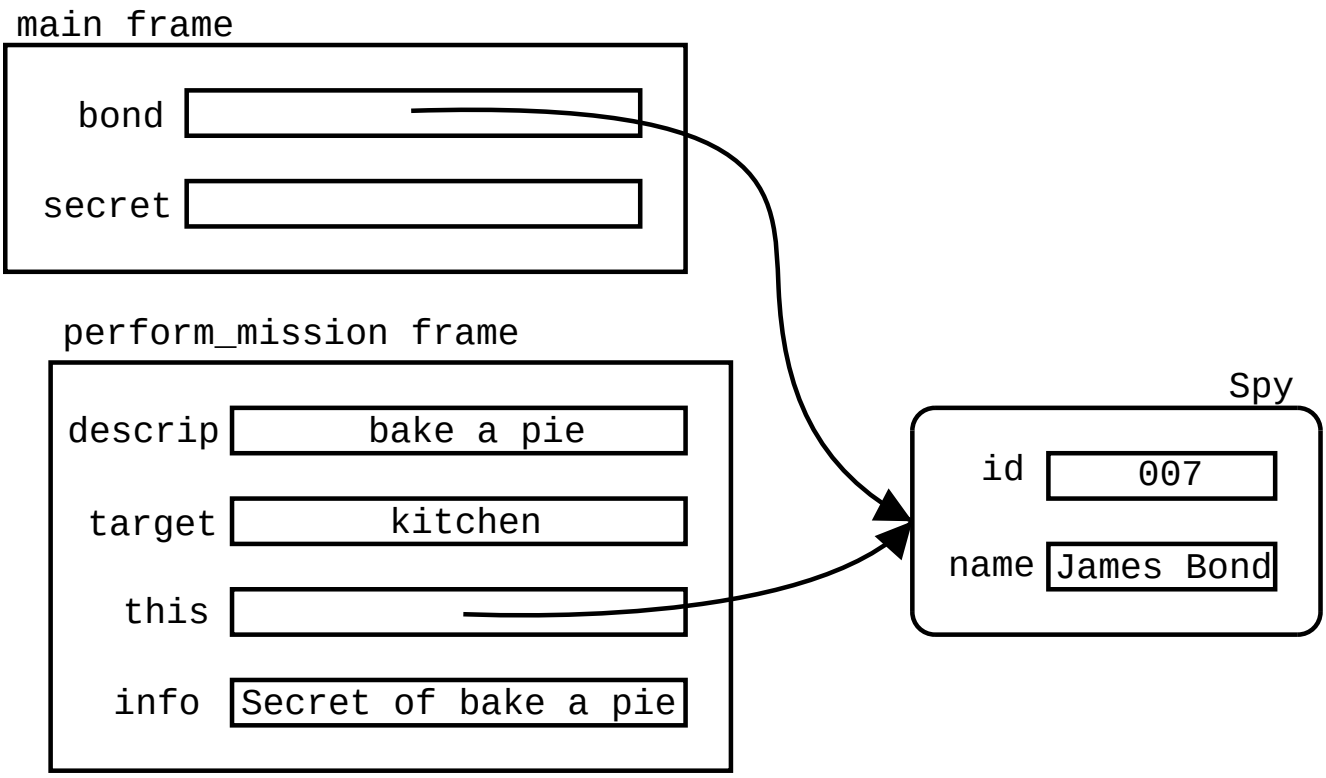
Method invocation



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;
  }
}
```

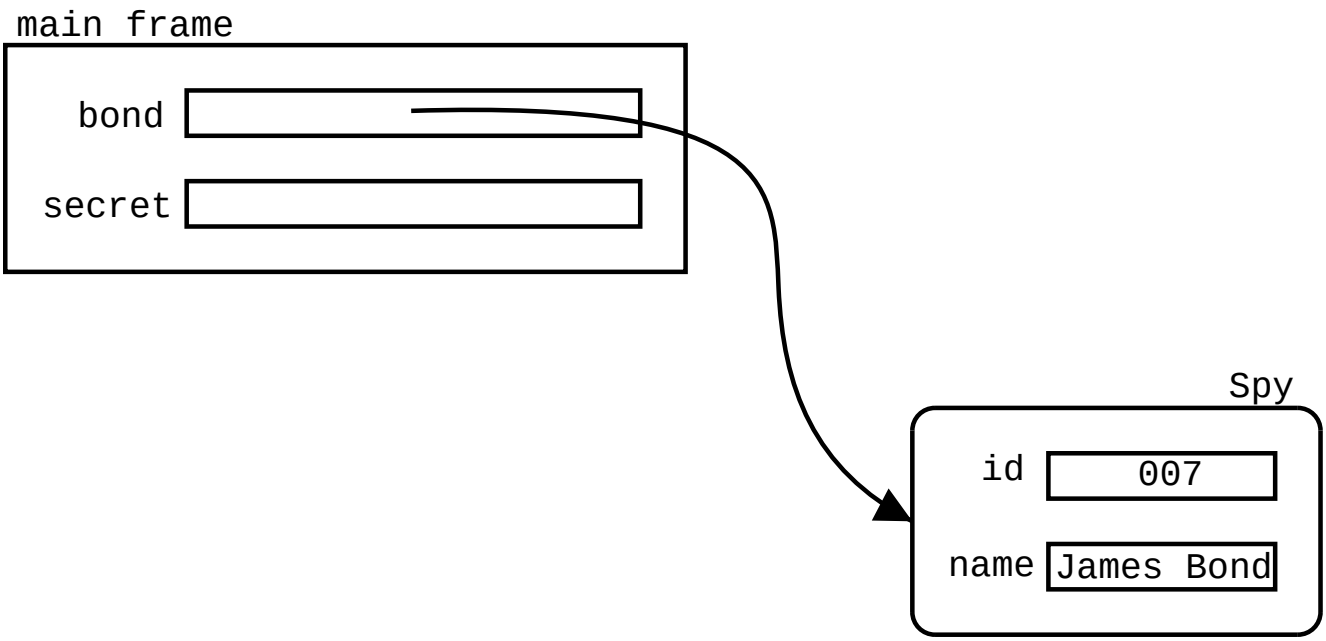
Method invocation



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;
  }
}
```

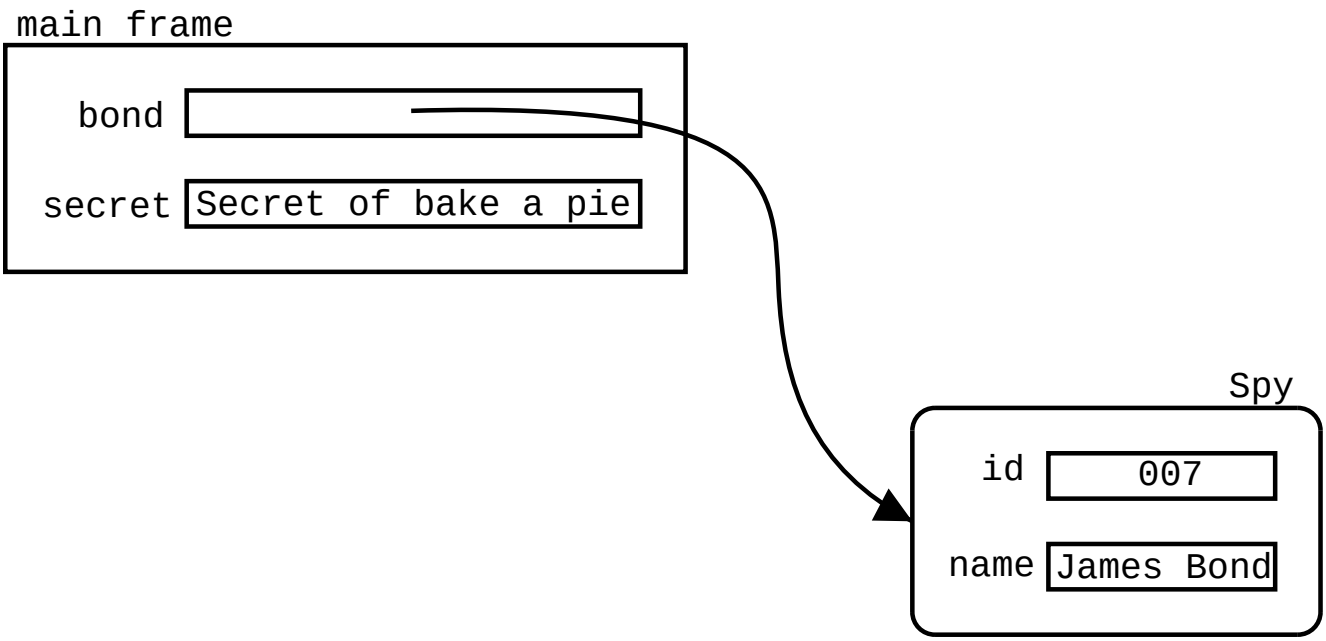
Method invocation



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



Objects are "first class citizens"

- A class is a data-type
- 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.

Objects as first class values

- Objects can be passed as parameters to a method

```
public class Theater {
  void play(Movie m)
  {
    m.print();
  }
}

public class MovieApplication {
  public static void main(String[] args)
  {
    Movie m1;
    Theater t = new Theater();
    m1 = new Movie(“Les Invasions barbares”,
                   “Denys Arcand”);
    t.play(m1);
  }
}
```

Objects as first class values

- Objects can be attibutes of other objects

```
public class Rabbit {
  void jump() { ... }
}
public class Cage {
  Rabbit my_rabbit;
  void put(Rabbit a)
  {
    my_rabbit = a;
  }
  Rabbit get()
  {
    return my_rabbit;
  }
}
```

...elsewhere...

```
Rabbit bugs = new Rabbit();
Cage c = new Cage();
c.put(bugs);
Rabbit wester = c.get();
```

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

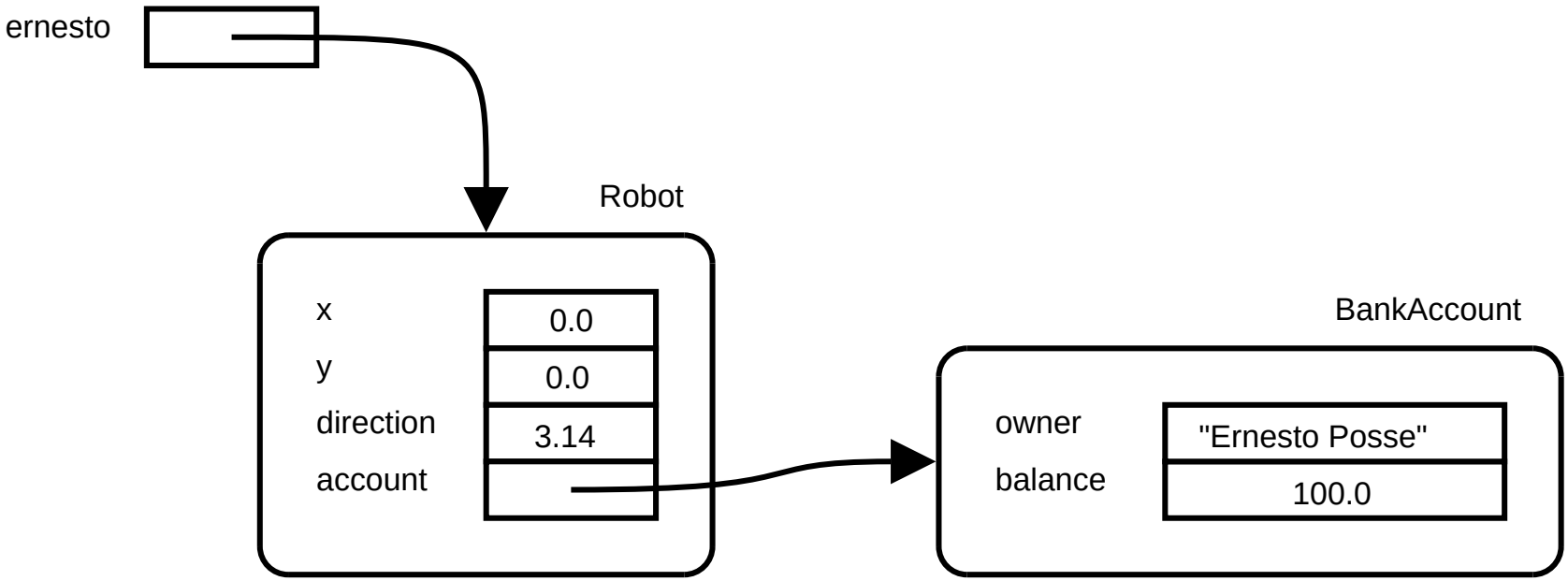
```
public class Robot
{
    double x, y, direction;
    BankAccount account;

    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



The end