# Recursion

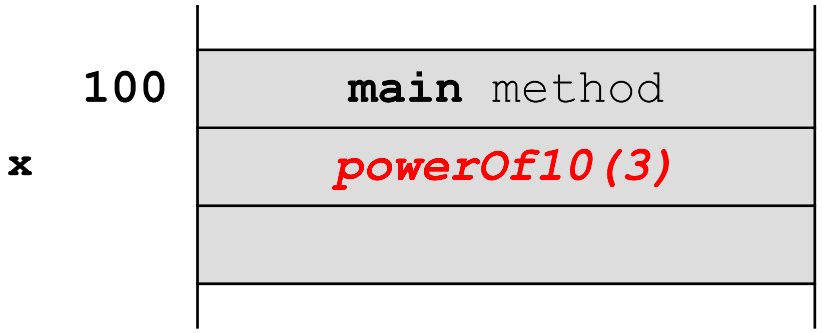
notes Chapter 8

## What Happens During Recursion?

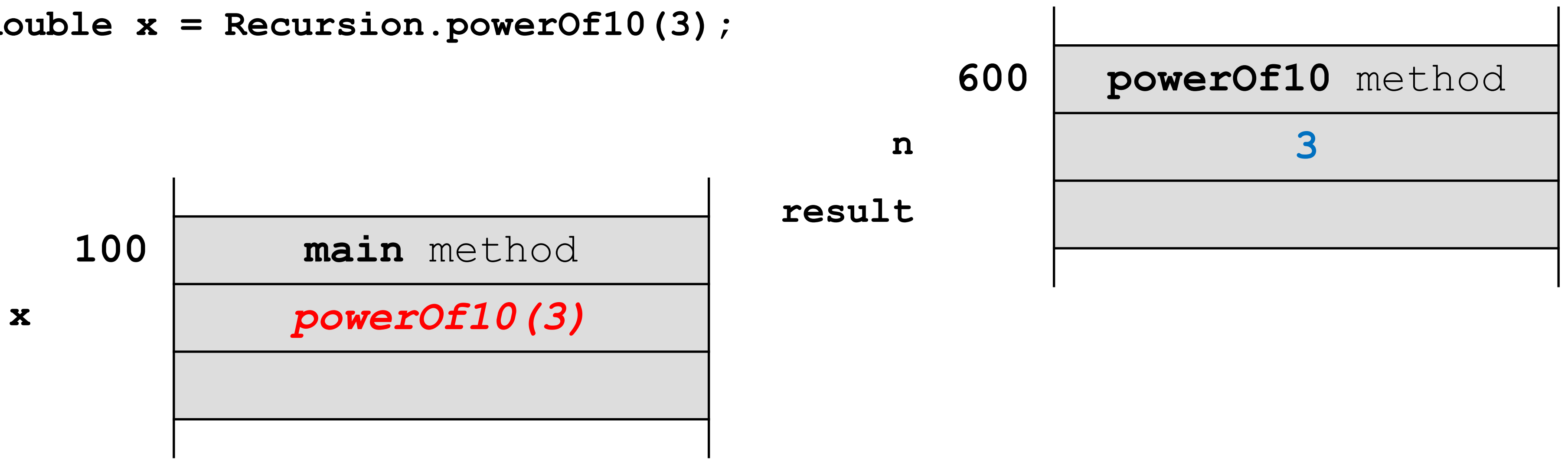
- a simplified model of what happens during a recursive method invocation is the following:
  - whenever a method is invoked that method runs in a *new* block of memory
    - when a method recursively invokes itself, a new block of memory is allocated for the newly invoked method to run in
- consider a slightly modified version of the **`powerOf10`** method

```
public static double powerOf10(int n) {
  double result;
  if (n < 0) {
    result = 1.0 / powerOf10(-n);
  }
  else if (n == 0) {
    result = 1.0;
  }
  else {
    result = 10 * powerOf10(n - 1);
  }
  return result;
}
```

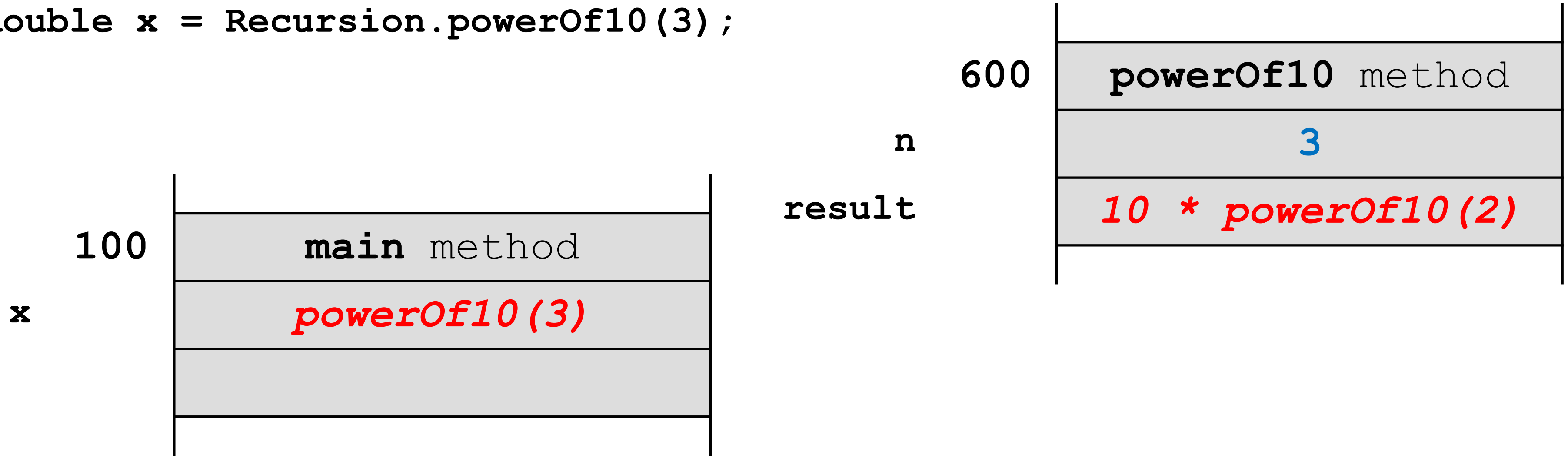
```
double x = Recursion.powerOf10(3);
```



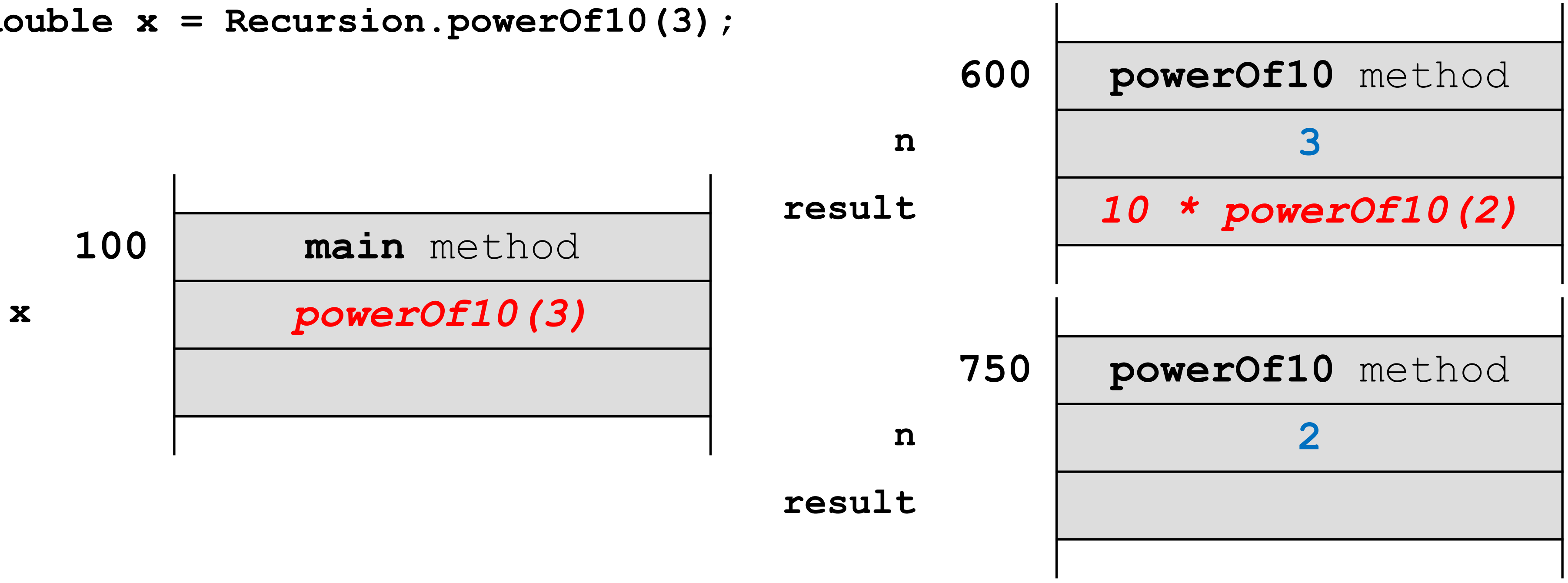
```
double x = Recursion.powerOf10(3);
```



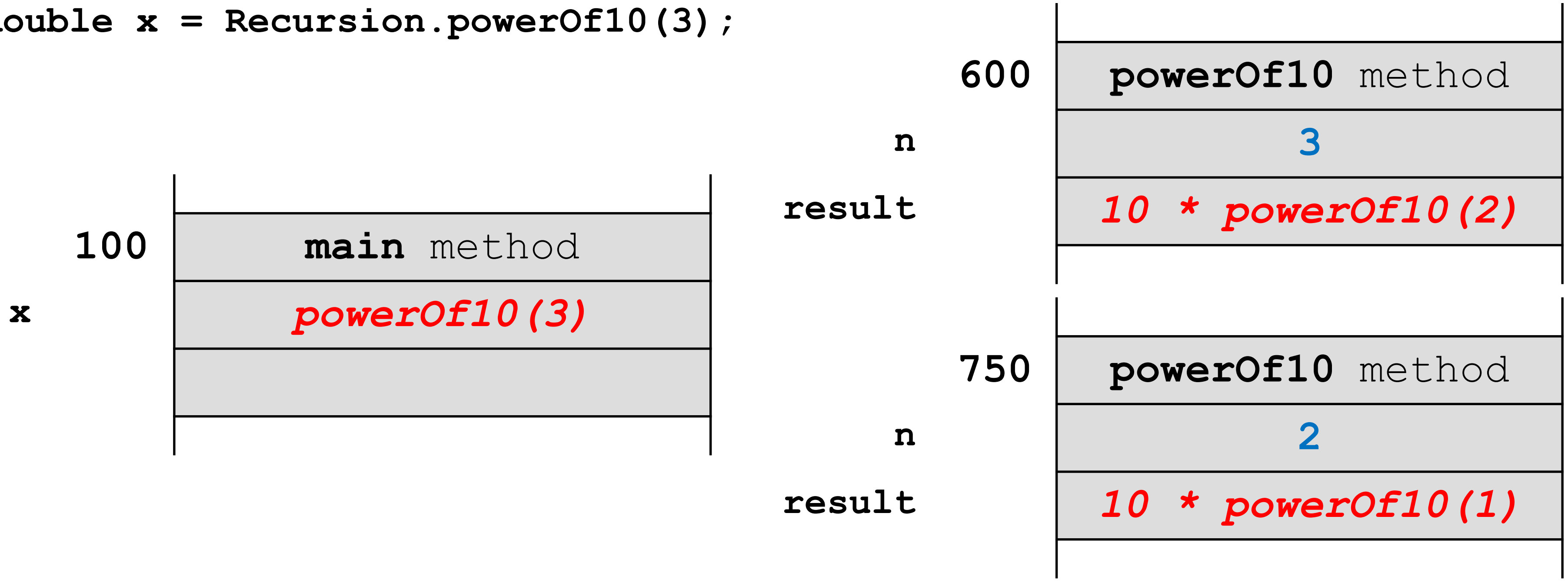
```
double x = Recursion.powerOf10(3);
```



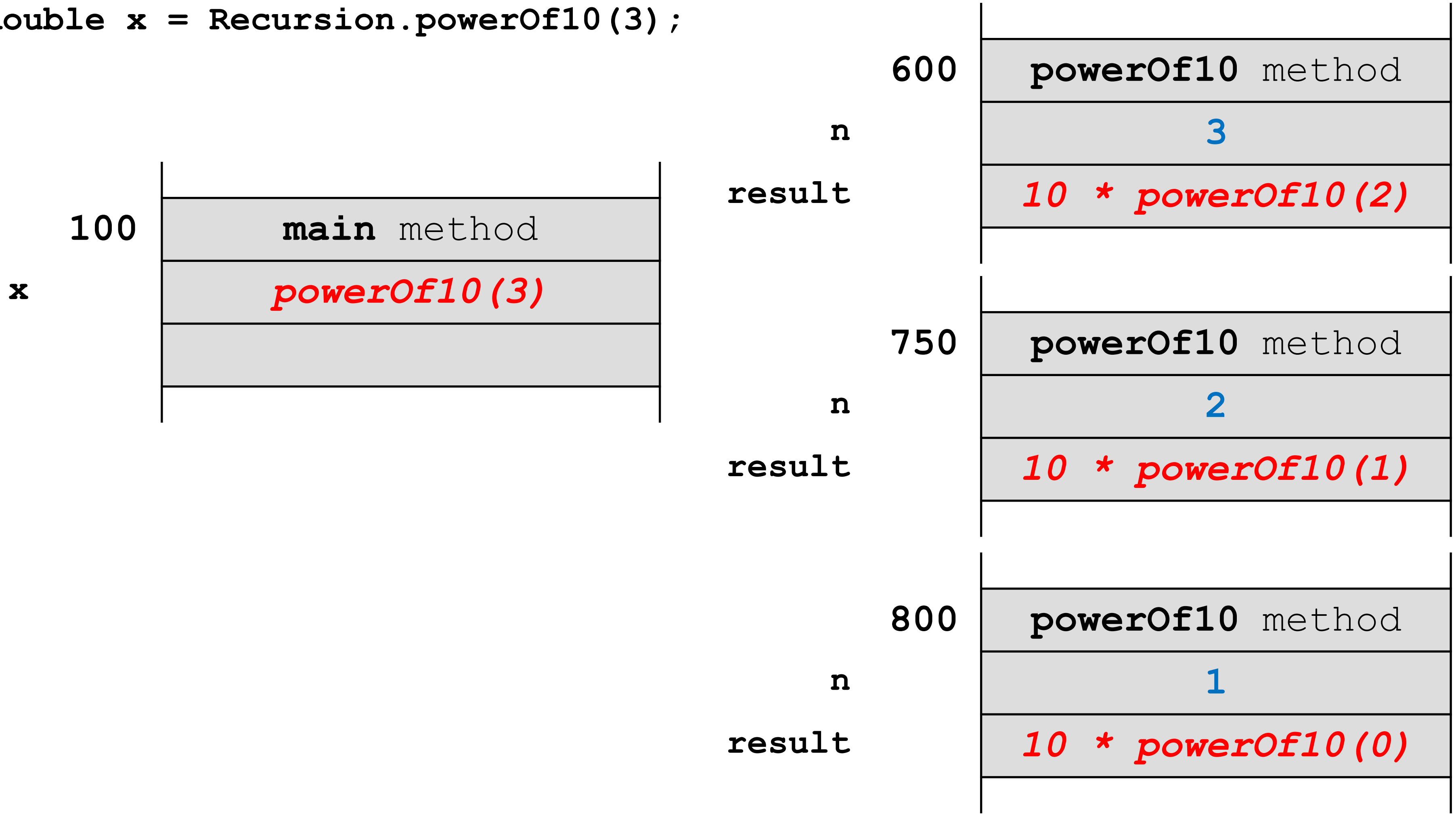
```
double x = Recursion.powerOf10(3);
```



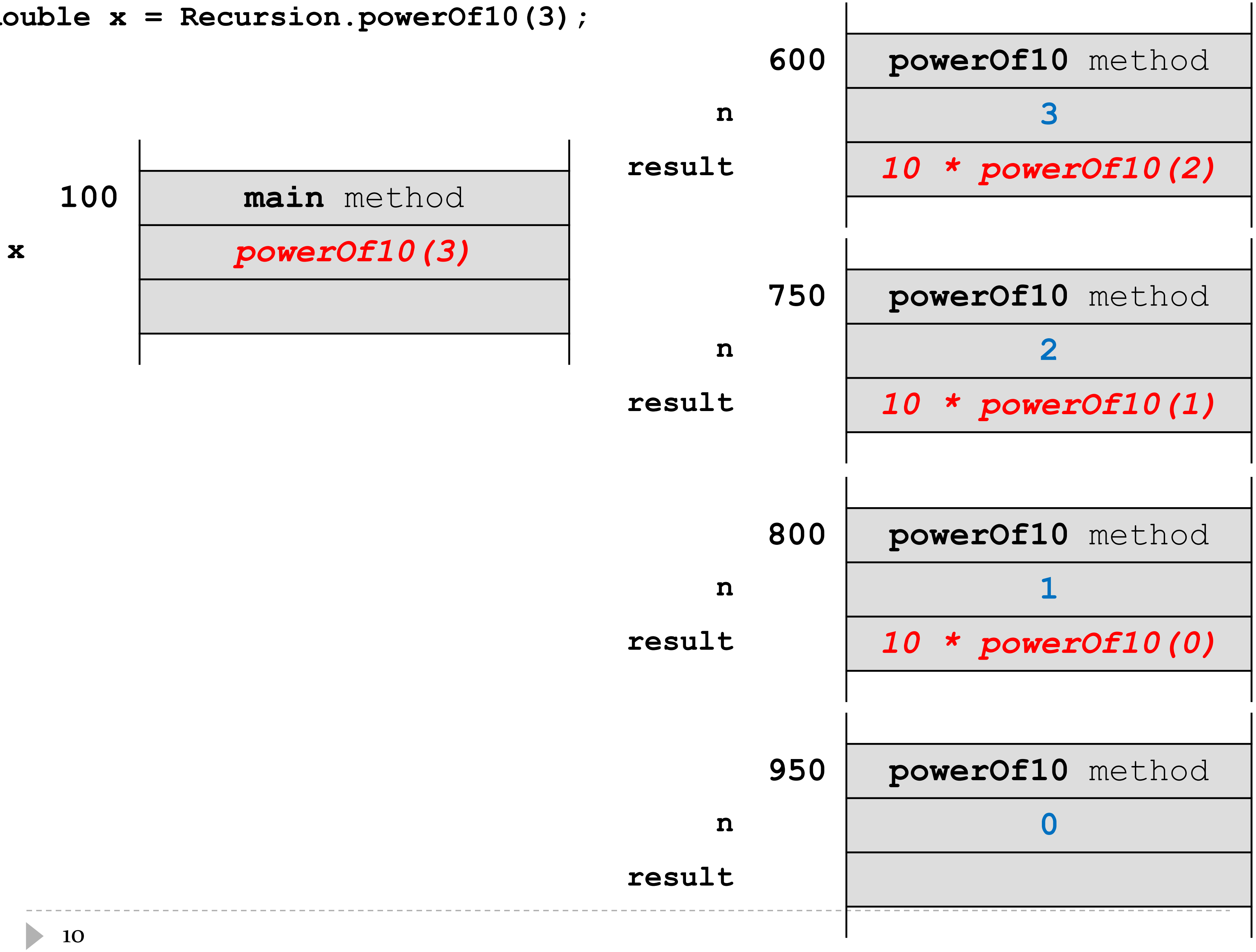
```
double x = Recursion.powerOf10(3);
```



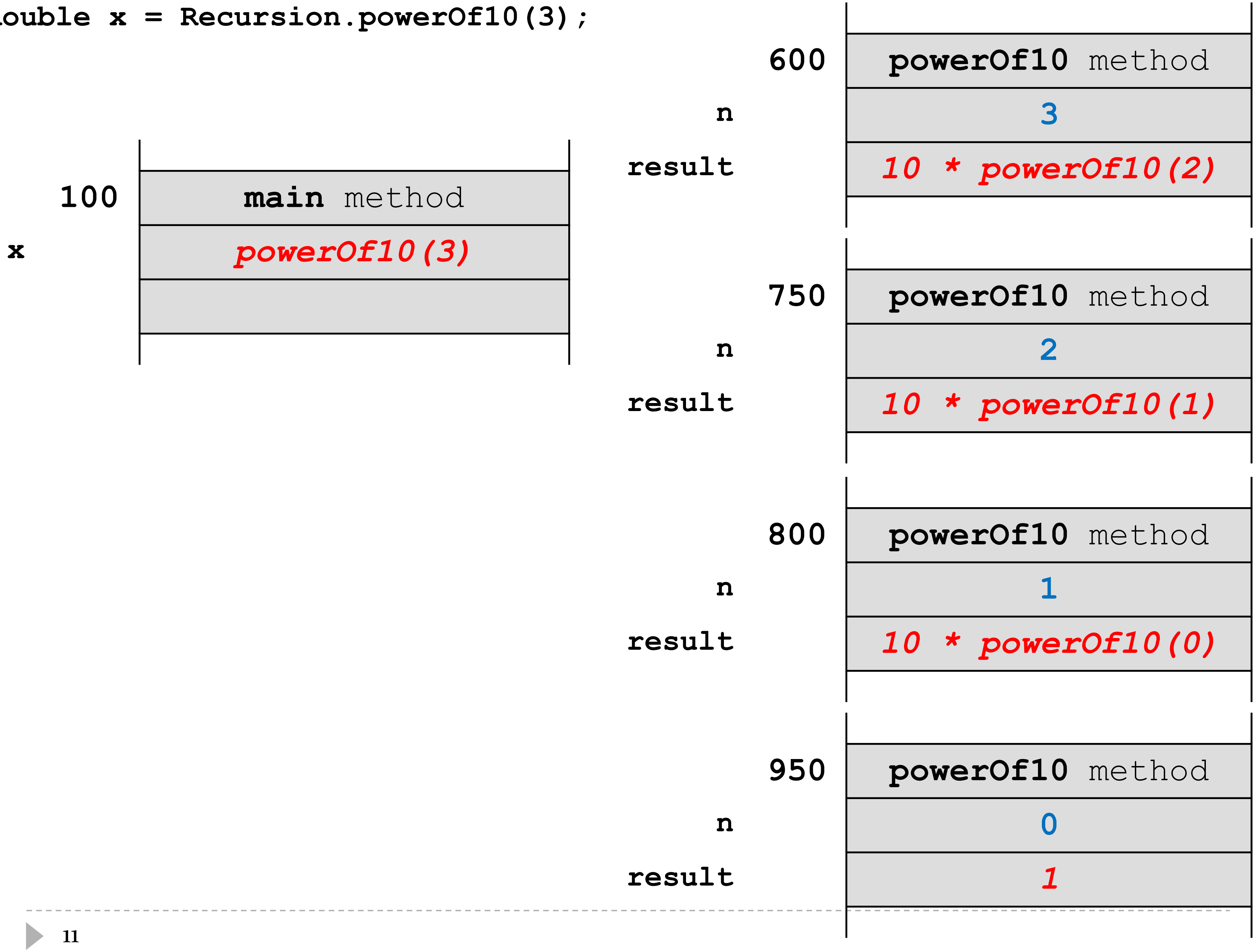
```
double x = Recursion.powerOf10(3);
```



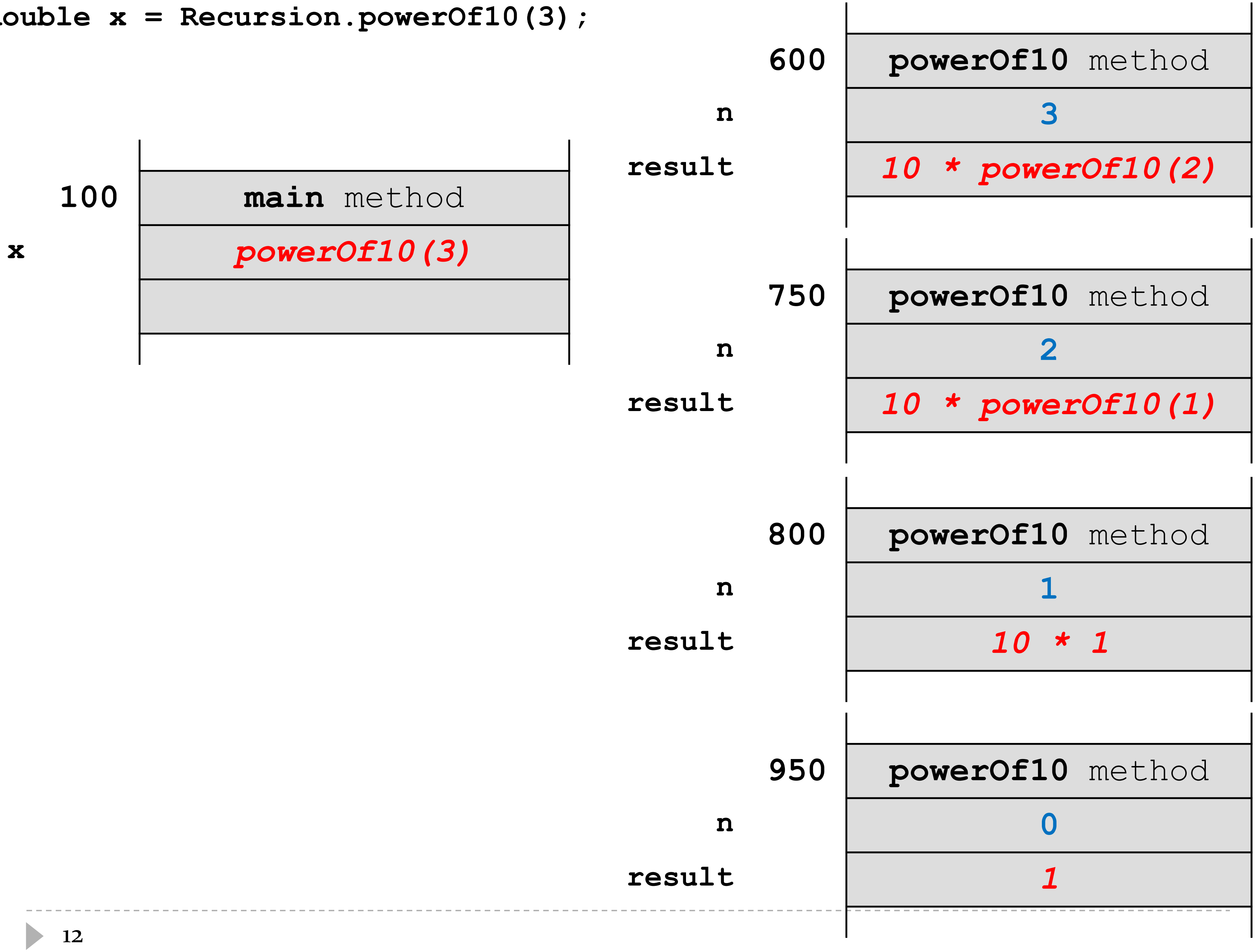
```
double x = Recursion.powerOf10(3);
```



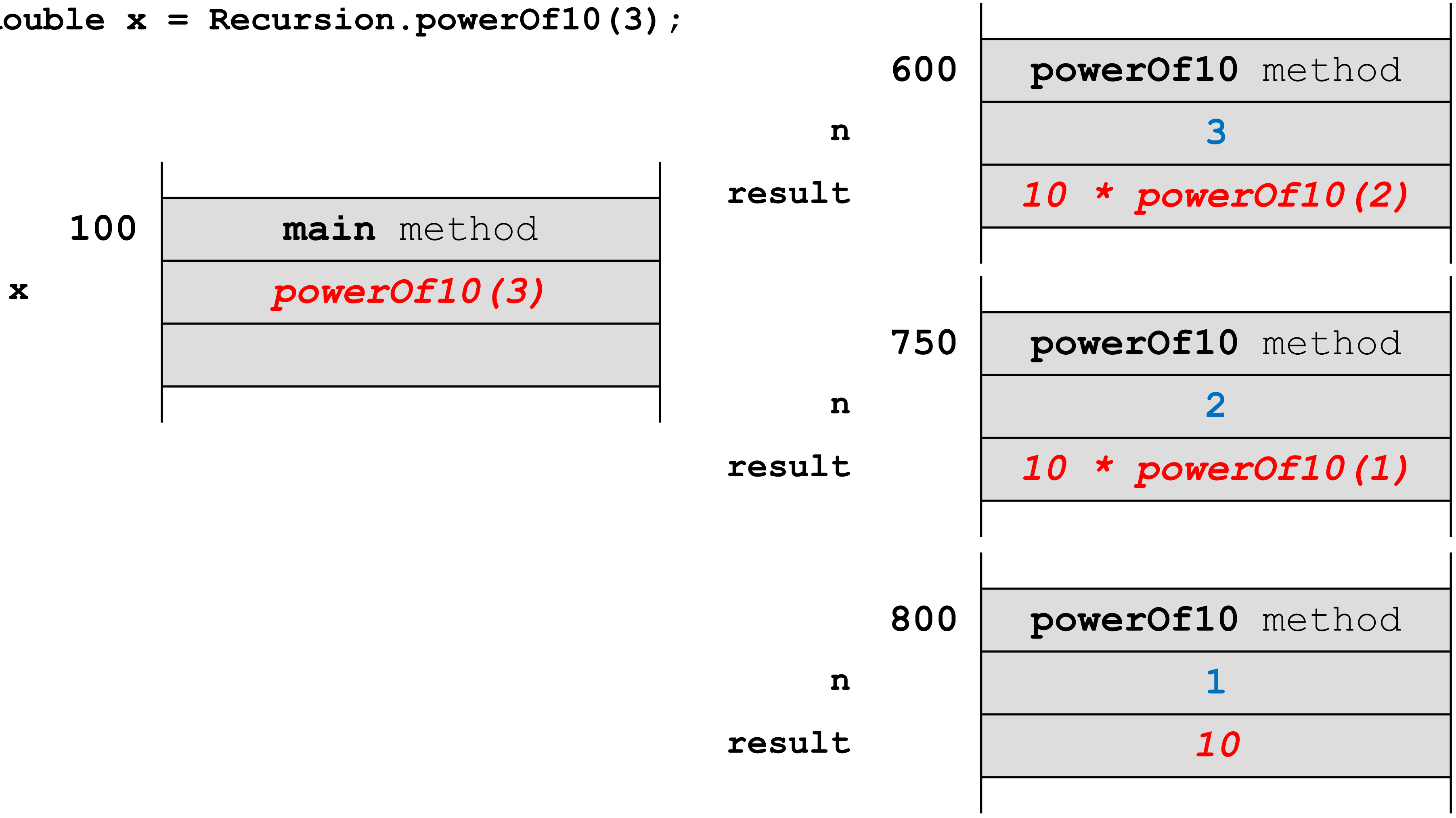
```
double x = Recursion.powerOf10(3);
```



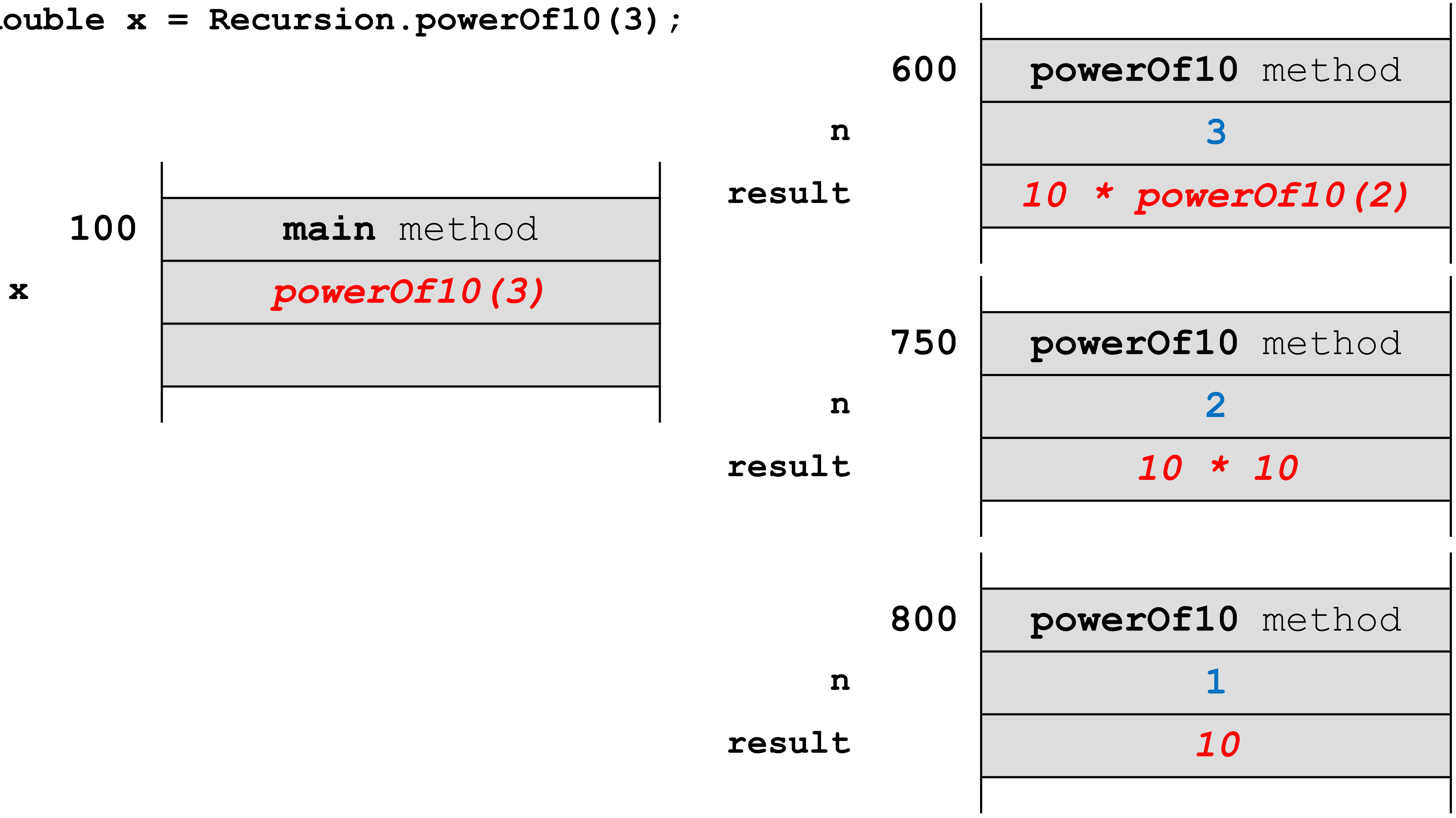
```
double x = Recursion.powerOf10(3);
```



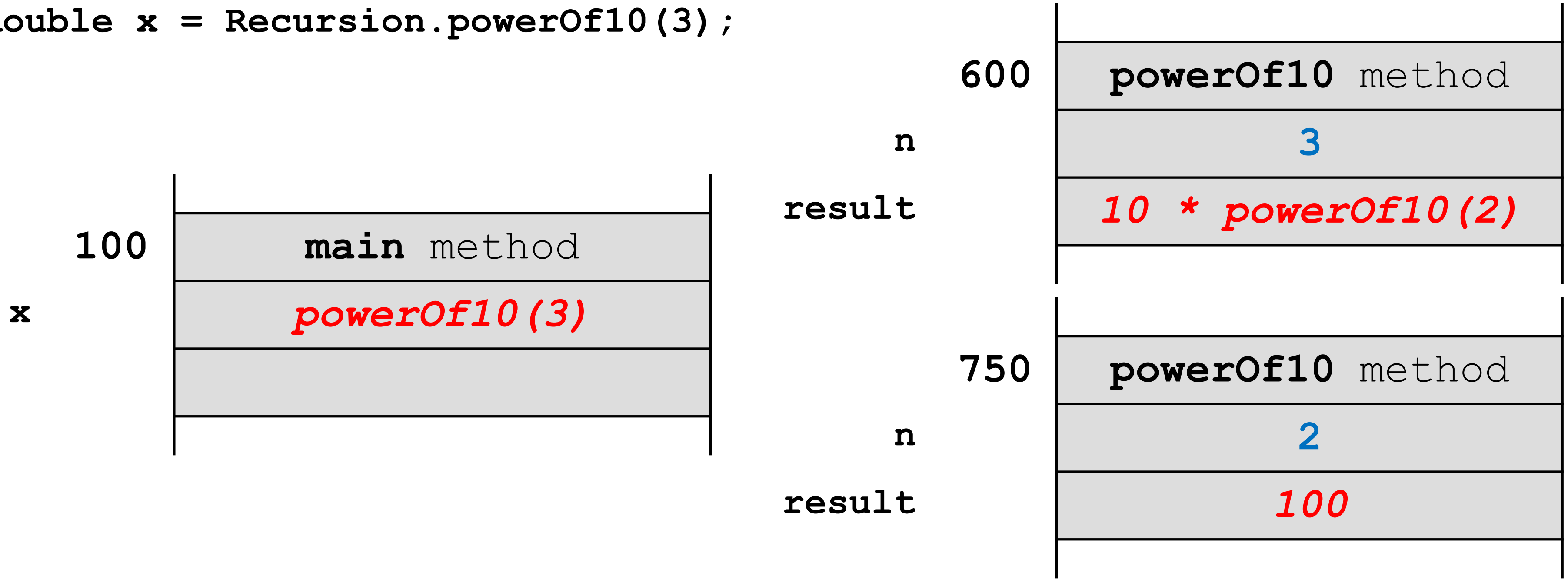
```
double x = Recursion.powerOf10(3);
```



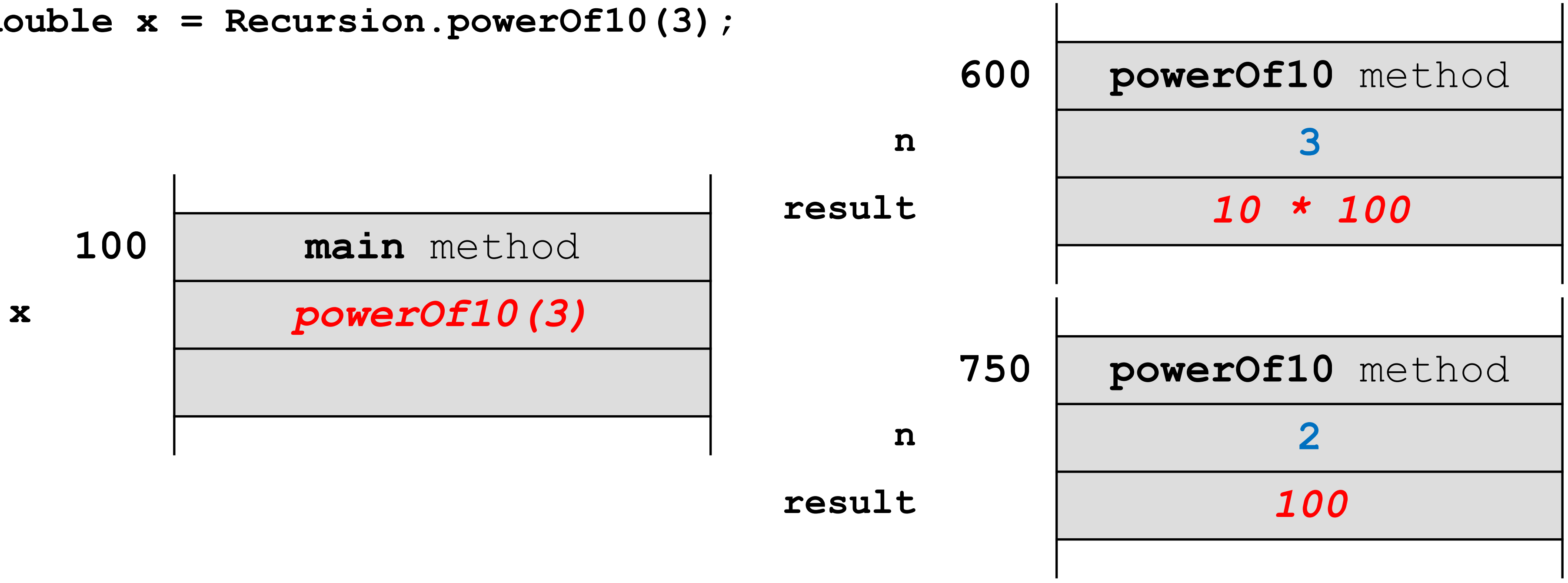
```
double x = Recursion.powerOf10(3);
```



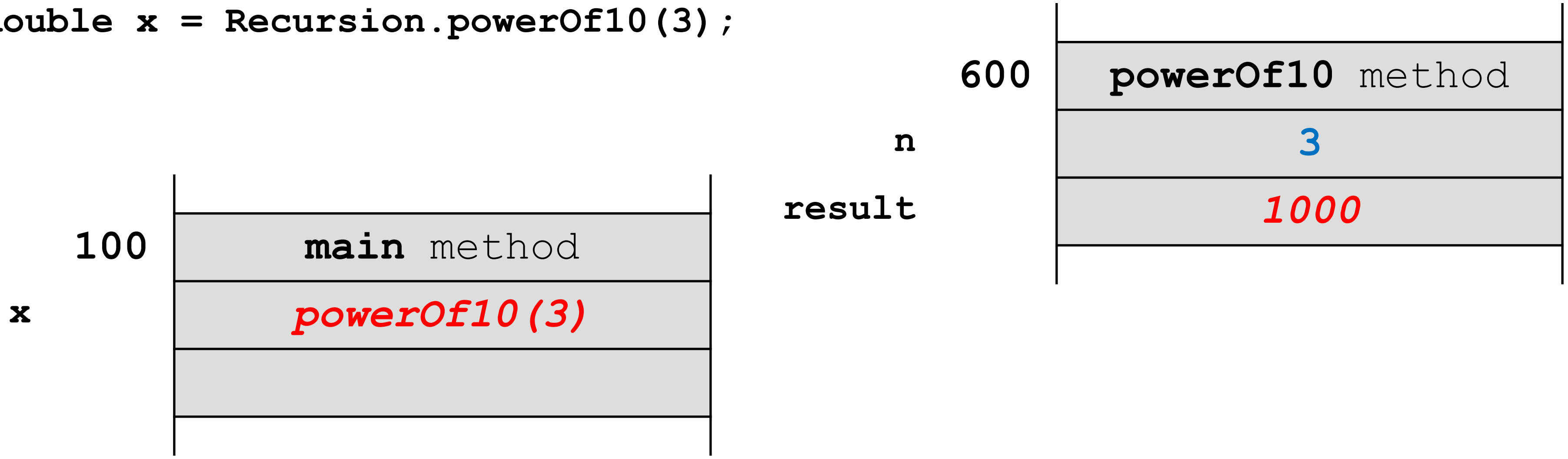
```
double x = Recursion.powerOf10(3);
```



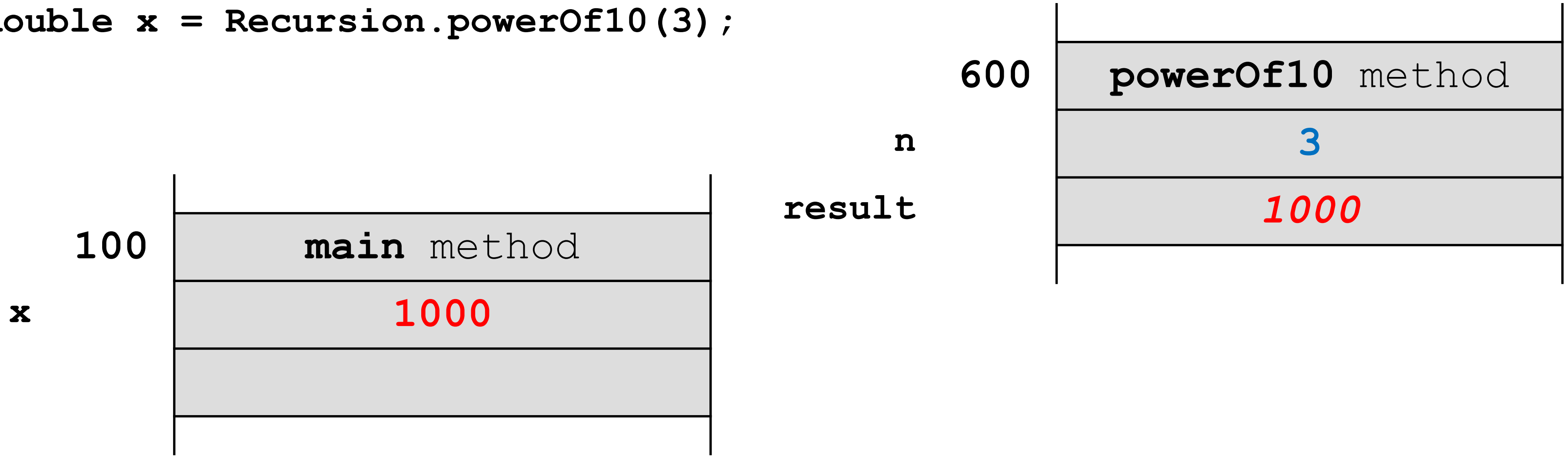
```
double x = Recursion.powerOf10(3);
```



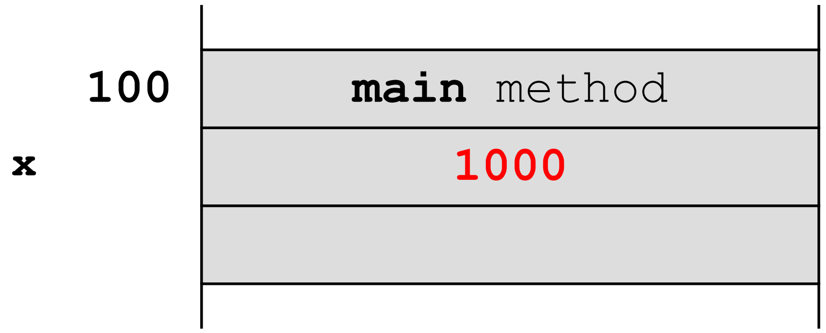
```
double x = Recursion.powerOf10(3);
```



```
double x = Recursion.powerOf10(3);
```



```
double x = Recursion.powerOf10(3);
```



## Recursion and Collections

- consider the problem of searching for an element in a list
- searching a list for a particular element can be performed by recursively examining the first element of the list
  - if the first element is the element we are searching for then we can return true
  - otherwise, we recursively search the sub-list starting at the next element
- example:
  search for `"X"` in the list `["Z", "Q", "B", "X", "J"]`

## Recursively Search a List

```
  contains("X", ["Z", "Q", "B", "X", "J"])

→ "X".equals("Z") == false
→ contains("X", ["Q", "B", "X", "J"])  recursive call

→ "X".equals("Q") == false
→ contains("X", ["B", "X", "J"])       recursive call

→ "X".equals("B") == false
→ contains("X", ["X", "J"])            recursive call

→ "X".equals("X") == true              done!
```

# Recursively Search a List

- base case(s)?

```
public class Day25 {

  public static <T> boolean contains(T element, List<T> t) {
    boolean result;
    if (t.size() == 0) {                          // base case
      result = false;
    }
    else if (t.get(0).equals(element)) {   // base case
      result = true;
    }



  }
}
```

## Recursively Search a List

- recursive call?

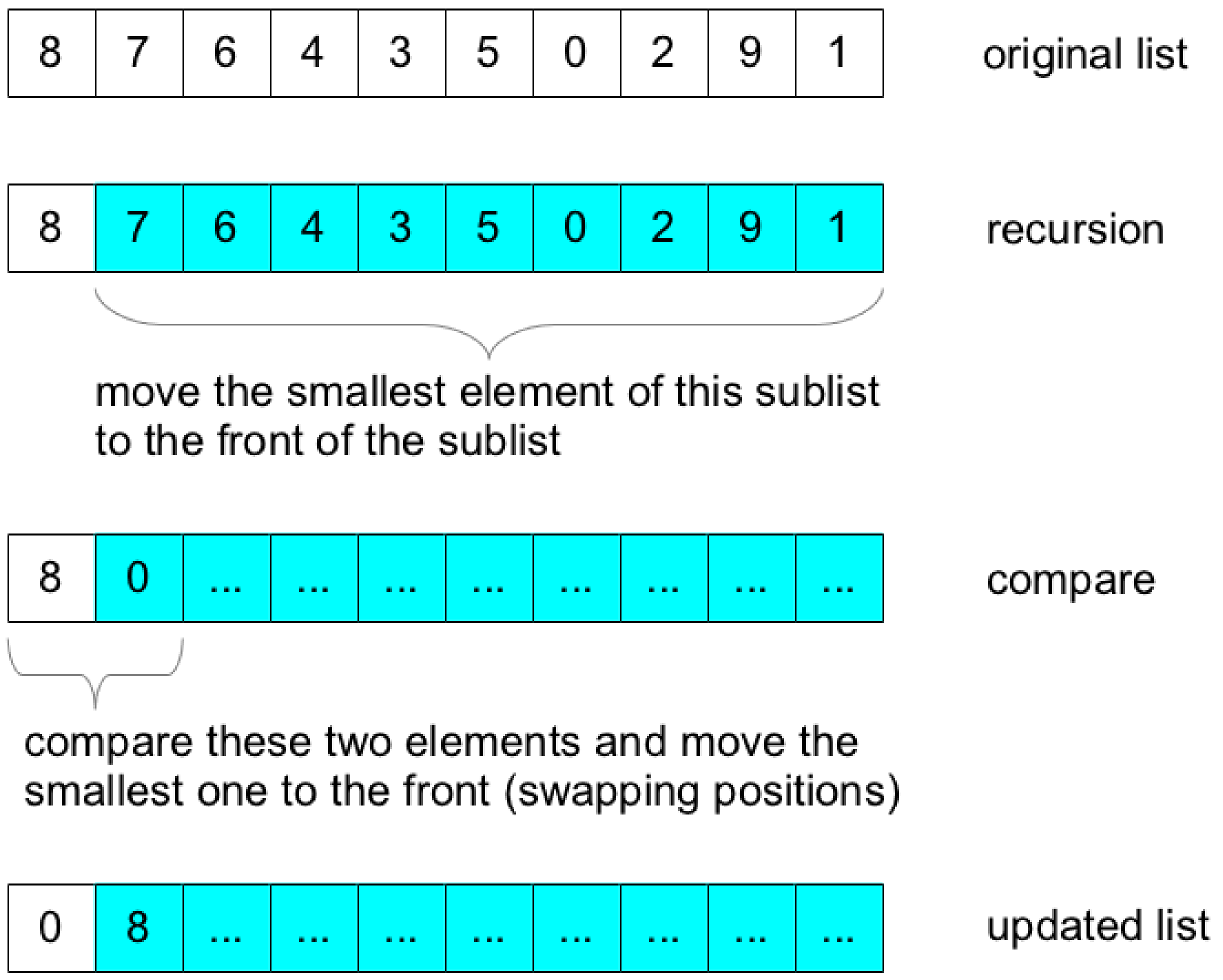
```
public class Day25 {

 public static <T> boolean contains(T element, List<T> t) {
  boolean result;
  if (t.size() == 0) {                    // base case
    result = false;
  }
  else if (t.get(0).equals(element)) {   // base case
    result = true;
  }
  else {                  // recursive call
    result = Day25.contains(element, t.subList(1, t.size()));
  }
  return result;
 }
}
```

## Recursion and Collections

- consider the problem of moving the smallest element in a list of integers to the front of the list

# Recursively Move Smallest to Front



8 7 6 4 3 5 0 2 9 1
original list
8 7 6 4 3 5 0 2 9 1
recursion
move the smallest element of this sublist to the front of the sublist
8 0 ... ... ... ... ... ... ... ...
compare
compare these two elements and move the smallest one to the front (swapping positions)
0 8 ... ... ... ... ... ... ... ...
updated list

## Recursively Move Smallest to Front

- base case?

## Recursively Move Smallest to Front

```
public class Day25 {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }







  }
}
```

# Recursively Move Smallest to Front

- recursive call?

## Recursively Move Smallest to Front

```
public class Day25 {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Day25.minToFront(t.subList(1, t.size()));





  }
}
```

http://docs.oracle.com/javase/7/docs/api/java/util/List.html#subList%28int,%20int%29

## Recursively Move Smallest to Front

- compare and update?

## Recursively Move Smallest to Front

```
public class Day25 {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Day25.minToFront(t.subList(1, t.size()));
    int first = t.get(0);
    int second = t.get(1);
    if (second < first) {
      t.set(0, second);
      t.set(1, first);
    }
  }
}
```