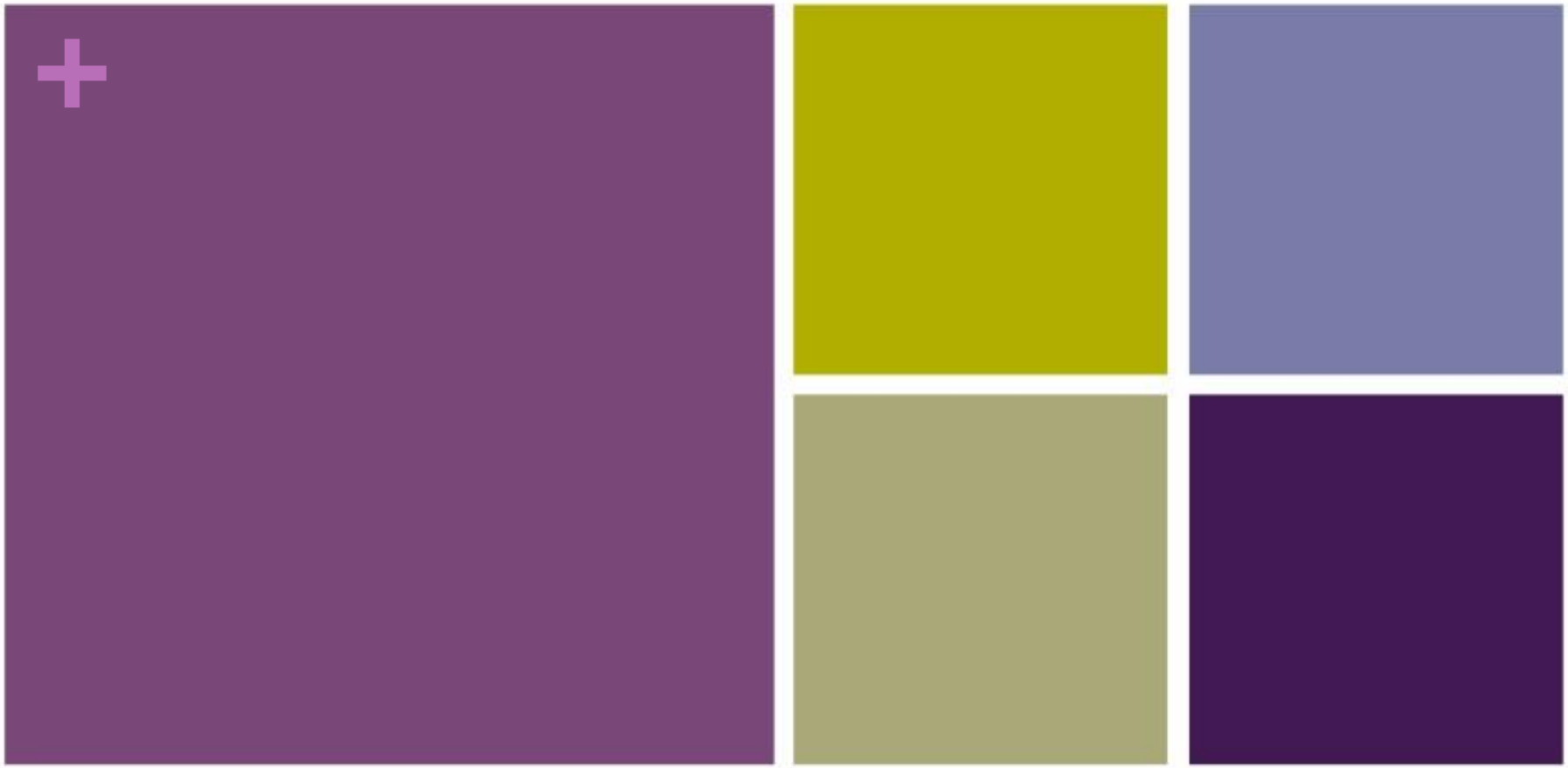


More on Repetition Structures & Methods

Introduction to Computer Science CSCI-UA.0101

Lecture 6

Agenda - Day 6

- Numerical Errors
- Nested Loops
- Introduction to Methods
- Defining a Method
- Calling a Method
- Value-returning Methods
- Void Methods
- Switch Selection Structure
- Examples

Numerical errors

Which is the output?

```
public class MyClass
{
   public static void main(String [] args)
   {
       double total = 0.6 + 0.3 + 0.1;
       if(total == 1.0)
       {
             System.out.println("total is equal to 1.0");
       }
       else
       {
             System.out.println("total is not equal to 1.0");
       }
   }
}
```

Programming Challenge

- Calculate the sum: 5.6 + 5.8 = ???
- Calculate the sum: 0.6 + 0.3 + 0.1 = ???
- So, what is the exact value?
- **NumericalErrors.java**

Programming Challenge

- Calculate the sum: 0.01 + 0.02 + 0.03 +…+ 0.99 +1.0
- You can also calculate the sum like this: 1.0 + 0.99 + …+ 0.03 + 0.02 + 0.01
- So, what is the exact value?
- **ForFloatCounter.java**

Nested Loops

Nested Loops

- A nested loop is a "loop inside another loop"
- We usually refer to nested loops in terms of their "outer" and "inner" loops. The outer loop is the first loop that is encountered in your program, and the inner loop is the loop that is nested inside of the outer loop. For example:

```
for (int outer = 0; outer < 10; outer++)
{
      // statement(s)

      for (int inner = 0; inner < 10; inner++)
      {
            // statement(s)
      }

      // statement(s)
}
```

Nested Loops

- In a nested loop configuration the inner loop will iterate through its full range one time for each iteration of the outer loop. For example:

```
for (int outer = 0; outer < 3; outer++)
{
      for (int inner = 0; inner < 3; inner++)
      {
            System.out.println(outer + " " + inner);
      }
}
```

QUESTIONS:

-How many times the System.out.println() statement will be executed?

-Which would be the output of the program?

Nested Loops

- In a nested loop configuration the inner loop will iterate through its full range one time for each iteration of the outer loop. For example:

```
for (int outer = 0; outer < 3; outer++)
{
      for (int inner = 0; inner < 3; inner++)
      {
            System.out.println(outer + " " + inner);
      }
}
```

OUTPUT:

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

Programming Challenge

- Find all prime numbers between 1 and 1,000
- Extension: print 10 items per line, formatted to 3 characters each (so everything lines up)
- **PrimeNumbers.java**

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Programming Challenge

- Programmatically generate a multiplication table for the number 9

Multiplication table for 9

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

- **MultiplicationTable.java**

Methods

Q1: What do we know about methods?

- Have you used methods (or functions) before this course?
- Have you used methods in this Java course?
- What are methods (or functions)?
- Why to use methods?

Q2: Identify methods in the following code

```
import java.util.Scanner;
public class MyClass
{
  public static void main(String [] args)
  {
      int num1;
      Scanner kb = new Scanner(System.in);
      num1 = kb.nextInt();
      System.out.println("Number: " + num1);
  }
}
```

Q2: Identify methods in the following code

```
import java.util.Scanner;
public class MyClass
{
  public static void main(String [] args)
  {
      int num1;
      Scanner kb = new Scanner(System.in);
      num1 = kb.nextInt();
      System.out.println("Number: " + num1);
  }
}
```

Why to use methods?

- Methods provide functionality to classes and are inherited by objects.
- Methods are commonly used to break a problem down into small manageable pieces. This is called *divide and conquer*.
- Methods simplify programs. If a specific task is performed in several places in the program, a method can be written once to perform that task, and then be executed anytime it is needed. This is known as *code reuse*.

`void` and Value-Returning Methods

- A `void` method is one that simply performs a task and then terminates.

```
System.out.println("Hi!");
```

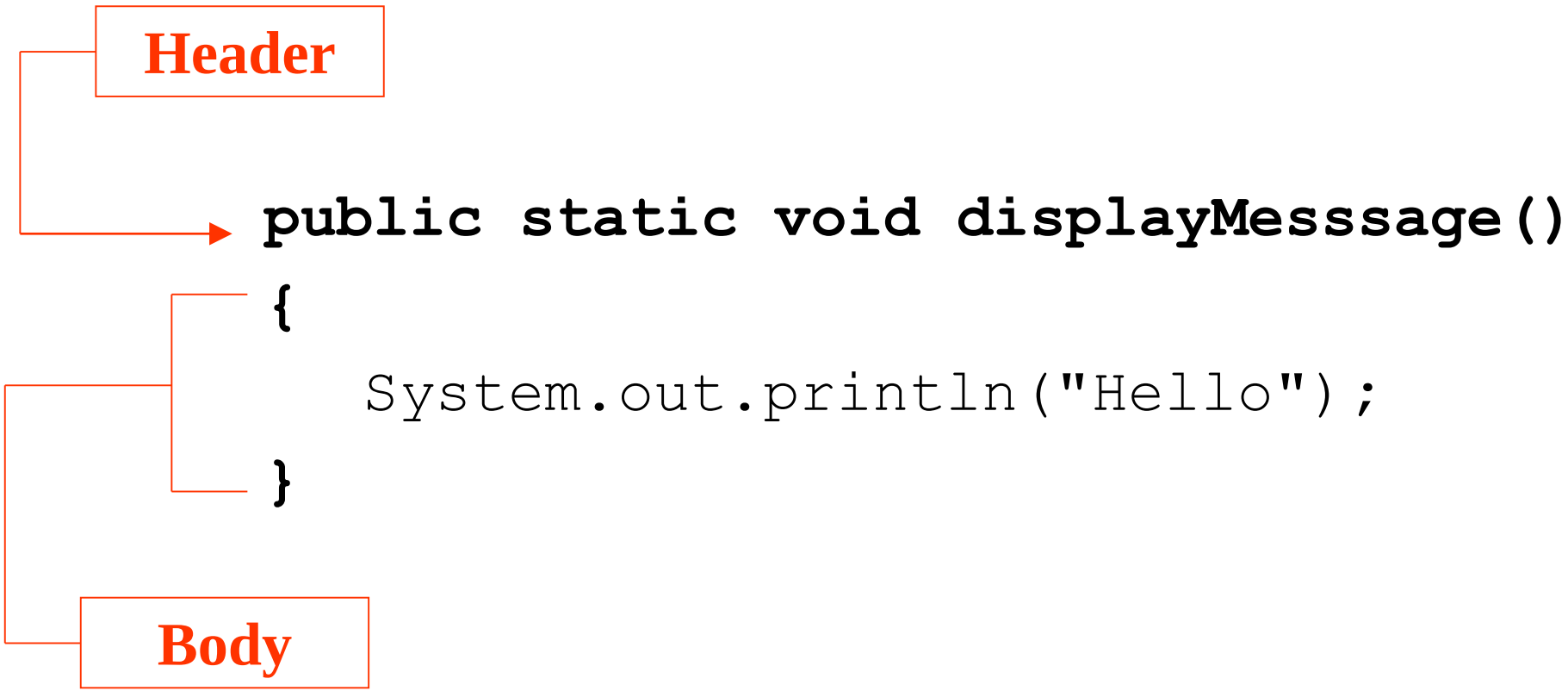
- A value-returning method not only performs a task, but also sends a value back to the code that called it.

```
int number = Integer.parseInt("700");
```

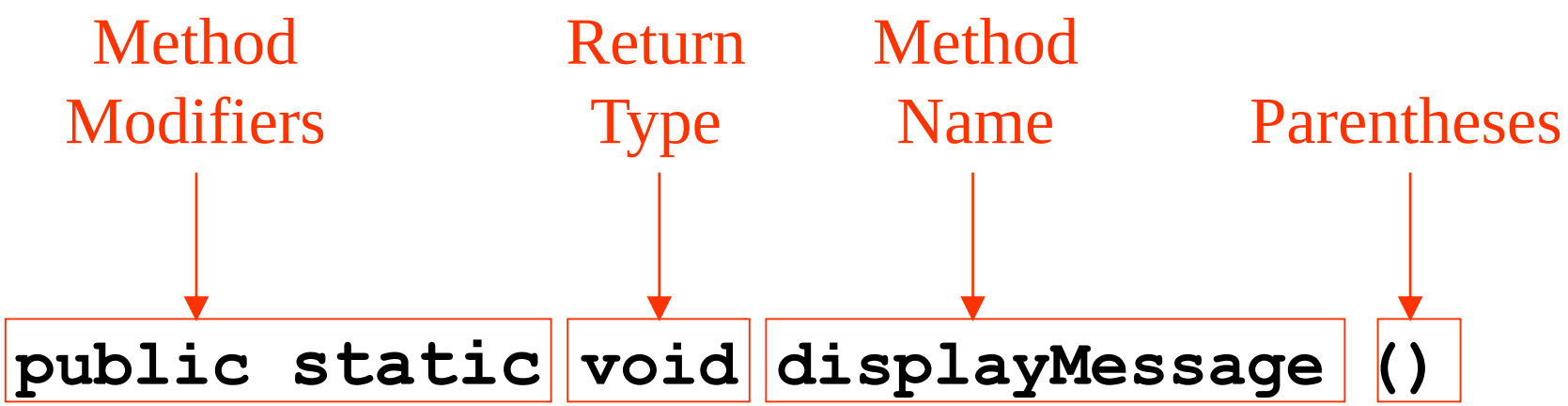
Defining a `void` Method

- To create a method, you must write a definition, which consists of a *header* and a *body*.
- The method header, which appears at the beginning of a method definition, lists several important things about the method, including the method's name.
- The method body is a collection of statements that are performed when the method is executed.

Method Definition: Header and Body



Structure of a Method Header




```
{
    System.out.println("Hello");
}
```

Structure of a Method Header

- Method modifiers
  - `public`—method is publicly available to code outside the class
  - `static`—method belongs to a class, not a specific object.
- Return type—`void` or the data type from a value-returning method
- Method name—name that is descriptive of what the method does
- Parentheses—contain nothing or a list of one or more variable declarations if the method is capable of receiving arguments.

Calling a Method

- A method executes when it is called.
- The `main` method is automatically called when a program starts, but other methods are executed by method call statements like:

```
displayMessage();
```

- Notice that the method modifiers and the `void` return type are not written in the method call statement. Those are only written in the method header.

Q3: Which is the output of the code below

```
public class MyClass
{
  public static void main(String [] args)
  {

  }
  public static void displayMesssage()
  {
     System.out.println("Hello");
  }
}
```

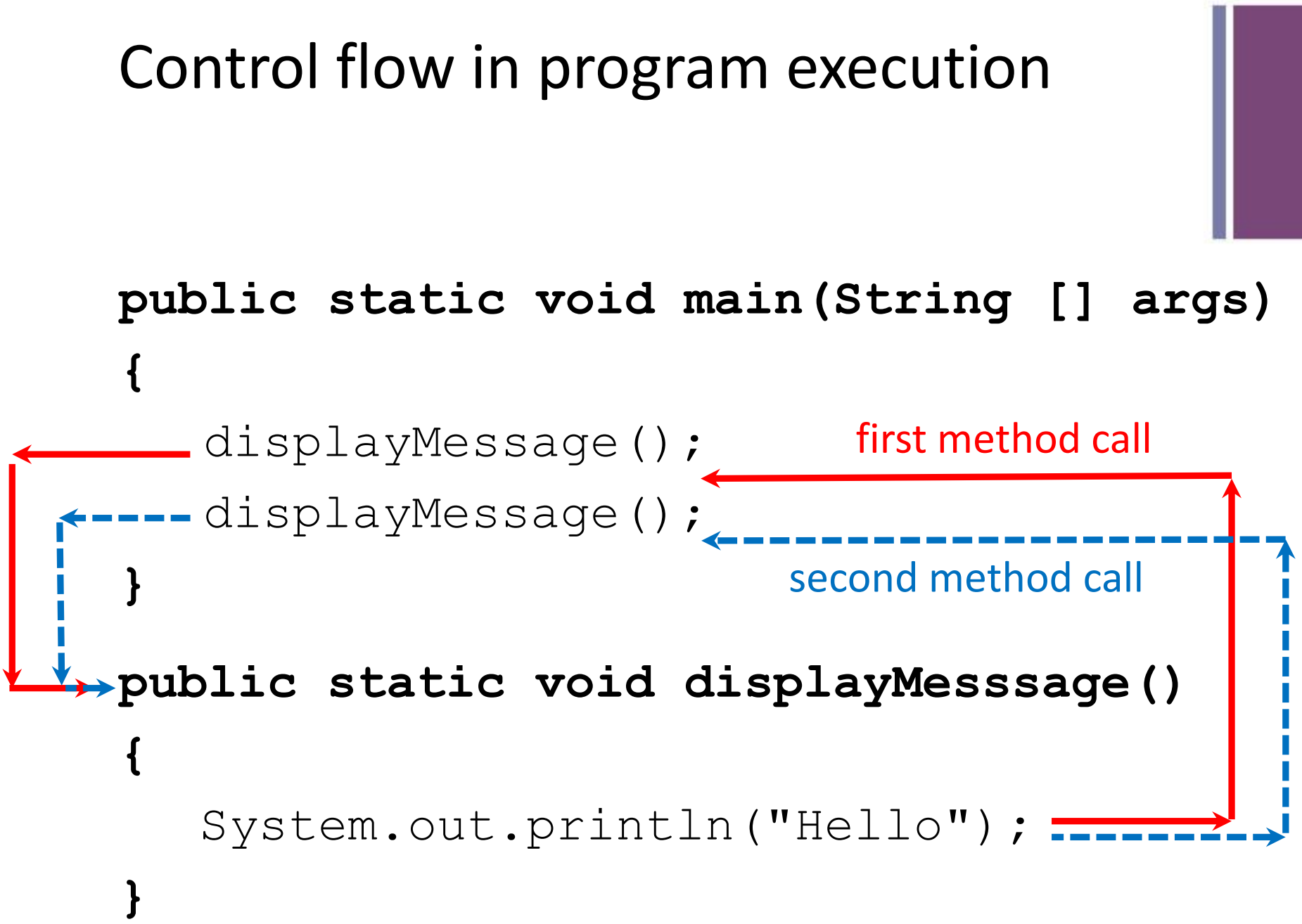
Q3: Which is the output of the code below

```
public class MyClass
{
  public static void main(String [] args)
  {
     displayMessage();
  }
  public static void displayMesssage()
  {
     System.out.println("Hello");
  }
}
```

Q3: Which is the output of the code below

```
public class MyClass
{
  public static void main(String [] args)
  {
     displayMessage();
     displayMessage();
  }
  public static void displayMesssage()
  {
     System.out.println("Hello");
  }
}
```

Control flow in program execution



Passing Arguments to a Method

- Values that are sent into a method are called arguments.

```
System.out.println("Hello");
number = Integer.parseInt(str);
```

- The data type of an argument in a method call must correspond to the variable declaration in the parentheses of the method declaration. The parameter is the variable that holds the value being passed into a method.
- By using parameter variables in your method declarations, you can design your own methods that accept data this way.

Passing a value to **displayValue()**

```
displayValue(5);     //called from main
```

The argument 5 is copied into the parameter variable **num**.

```
public static void displayValue(int num)
{
  System.out.println("The value is " + num);
}
```

The method will display **The value is 5**

Argument and Parameter Data Type Compatibility

- When you pass an argument to a method, be sure that the argument's data type is compatible with the parameter variable's data type.

- Java will automatically perform widening conversions, but narrowing conversions will cause a compiler error.

```
double d = 1.0;
displayValue(d);
```

Error! Can't convert `double` to `int`

Passing Multiple Arguments

The argument 5 is copied into the **num1** parameter.

The argument 10 is copied into the **num2** parameter.

```
showSum(5, 10); //called from main
```

NOTE: Order matters!

```
public static void showSum(double num1, double num2)
{
   double sum;   //to hold the sum
   sum = num1 + num2;
   System.out.println("The sum is " + sum);
}
```

Arguments are Passed by Value

- In Java, all arguments of the primitive data types are *passed by value*, which means that only a copy of an argument's value is passed into a parameter variable.

- A method's parameter variables are separate and distinct from the arguments that are listed inside the parentheses of a method call.

- If a parameter variable is changed inside a method, it has no affect on the original argument.

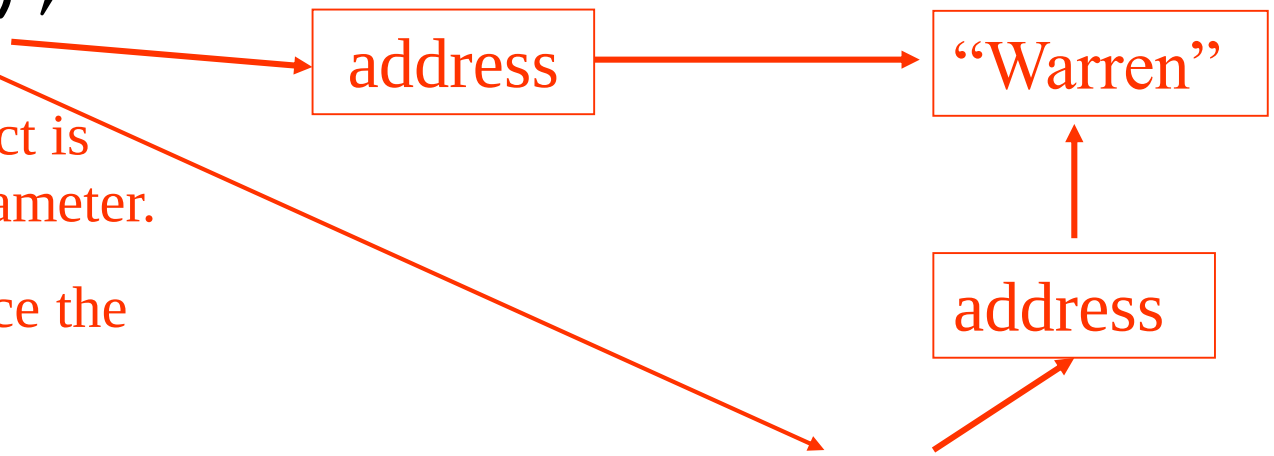
Passing Object References to a Method

- Recall that a class type variable does not hold the actual data item that is associated with it, but holds the memory address of the object. A variable associated with an object is called a reference variable.

- When an object such as a `String` is passed as an argument, it is actually a reference to the object that is passed.

Passing a Reference as an Argument

```
String name = "Warren";
showLength(name);
```



-The address of the object is copied into the **str** parameter.

-**name** and **str** reference the string object "Warren"

```
public static void showLength(String str)
{
  System.out.println(str + " is " + str.length()+
                     " characters long.");
}
```

Programming Challenge

- Write a method called helloWorld that prints out the string "Hello, World!" when called. This method should return nothing to the caller.

- Extend your method to accept a String as an argument, and augment your output to include that string in the output text. For example:

  Hello, World, Craig!

- **Greetings.java**

Returning a Value from a Method

- Data can be passed into a method by way of the parameter variables. Data may also be returned from a method, back to the statement that called it.

```
int num = Integer.parseInt("700");
```

- The string “700” is passed into the `parseInt` method.
- The `int` value 700 is returned from the method and assigned to the `num` variable.

Defining a Value-Returning Method

```
public static int sum(int num1, int num2)
{
  int result;
  result = num1 + num2;
  return result;
}
```

return type

This expression must be of the same data type as the return type

The `return` statement causes the method to end execution and it returns a value back to the statement that called the method.

Calling a Value-Returning Method

```
int value1 = 20, value2 = 40;
total = sum(value1, value2); //from main
```

20

40

```
  public static int sum(int num1, int num2)
  {
    int result;
    result = num1 + num2;
    return result;
  }
```

60

Q4: Read/Analyze the following code

```
public class MyClass
{
   public static void main(String [] args)
   {
        int value = 20;
        if(isValid(value))
               System.out.println("The value is within range");
        else
               System.out.println("The value is out of range");
   }
    public static boolean isValid(int number)
    {
        boolean status;
        if(number >= 1 && number <= 100)
           status = true;
        else
           status = false;
        return status;
    }
}
```

1) Name the methods used in the code?
2) Is local variable `number` known to main?
3) Which is the type of the value returned by isValid?
4) Which is the output produced by the code?

Returning a Reference to a `String` Object

```
customerName = fullName("John", "Martin");

public static String fullName(String first, String last)
{
    String name;
    name = first + " " + last;
    return name;
}
```

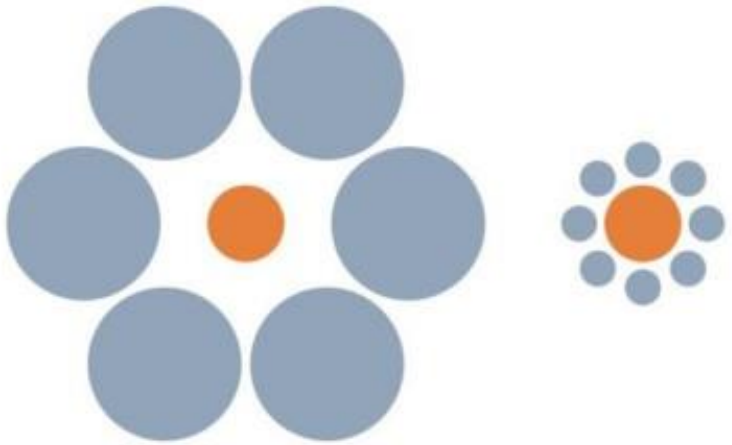
address

"John Martin"

Local variable `name` holds the reference to the object. The return statement sends a copy of the reference back to the call statement and it is stored in `customerName`.

Programming Challenge

- Write a method called "max" that accepts two integers as parameters.
- Return the larger of the two integers to the caller of the method.
- **CalcMax.java**



Programming Challenge

- Write a method called "sumRange" that accepts two integers as parameters.
- Sum up the numbers in the range provided and return the result to the caller.
- **Range.java**

Summary: Problem Solving with Methods

- A large, complex problem can be solved by breaking it down in methods.
- The process of breaking a problem down into smaller pieces is called *functional decomposition (divide and conquer)*.
- To use (call) a method you need to write a statement in **`main`** that calls the method.
- To define a method you need to specify the *header* (arguments, return value) and the *body* of the method (what the method does or computes).

Memory Management: Stack Memory

The Stack

- Every time a method is called Java creates an "activation record" that stores information about the method call, such as the name of the method and any variables that are involved in the call.
- The activation record is placed into an area of memory called the "stack". The caller's activation record is also in memory on the stack.
- The stack operates as a "first in, last out" construct. This means that activation records are placed on top of the stack when a method is called. When the method call terminates the activation record is removed and control is passed back to the caller's activation record.

The Stack

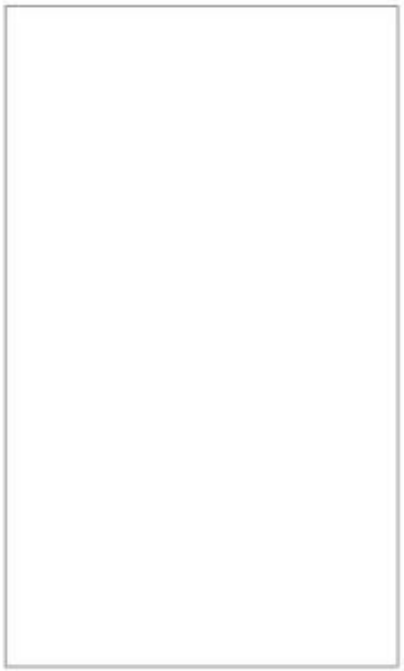
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

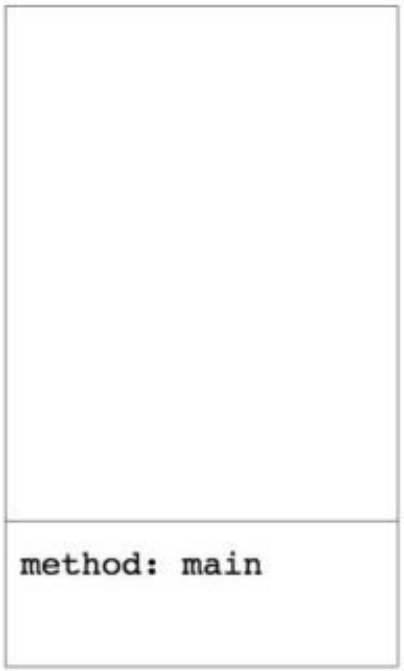
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

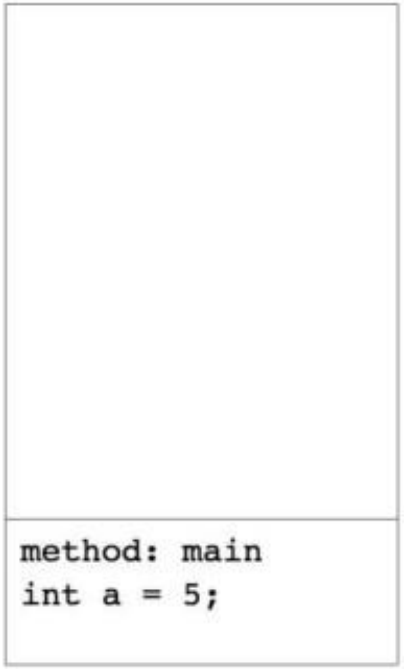
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

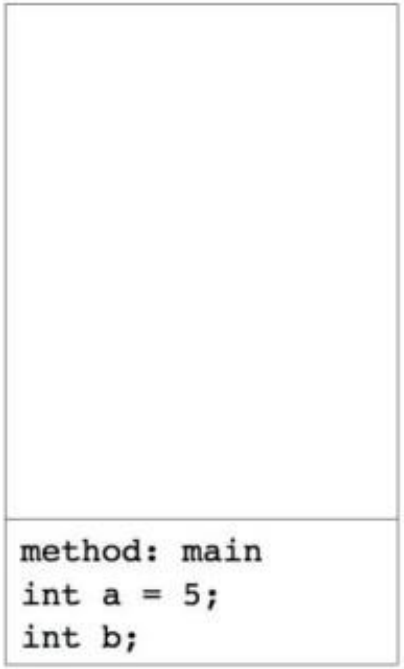
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

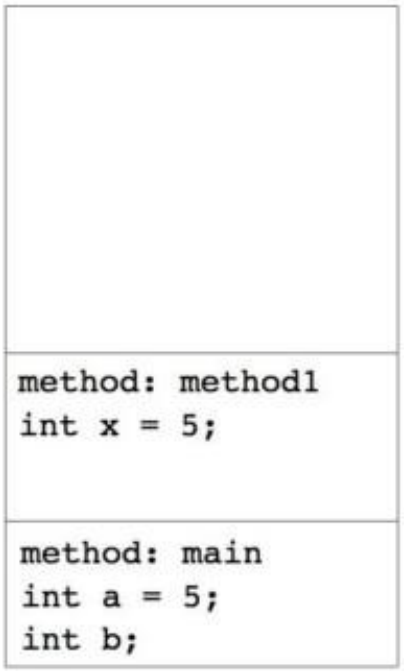
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

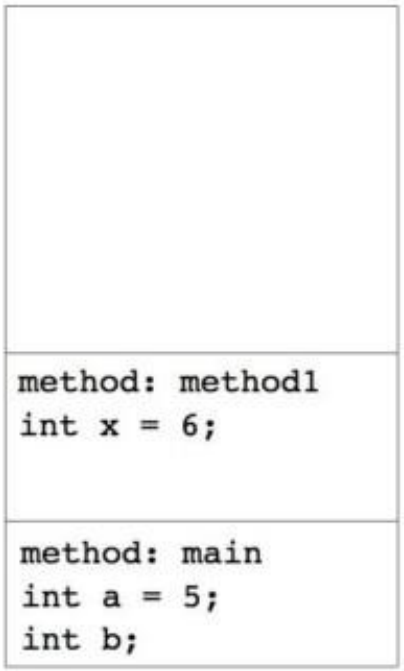
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

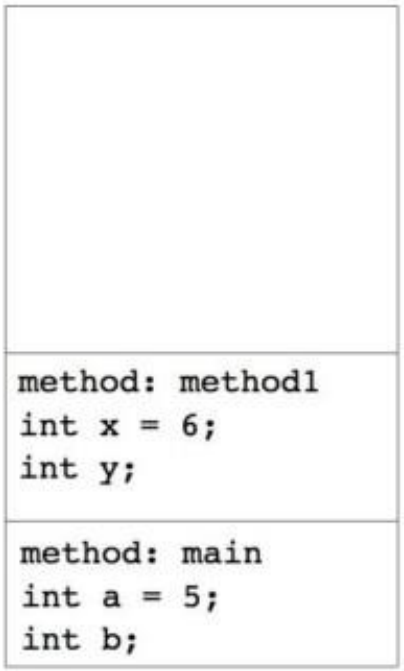
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

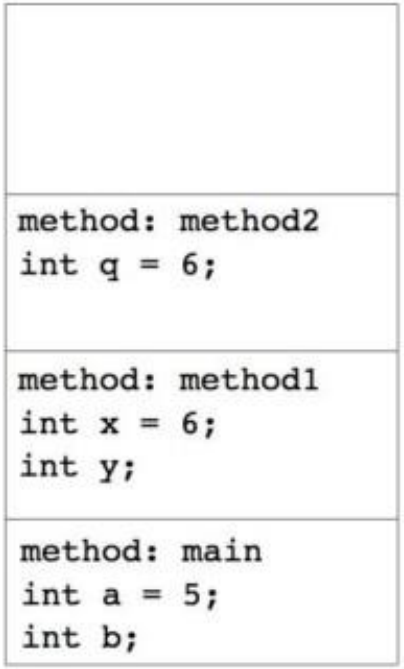
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

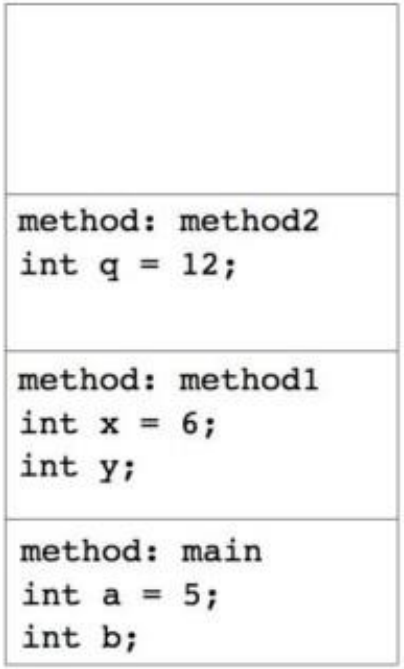
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

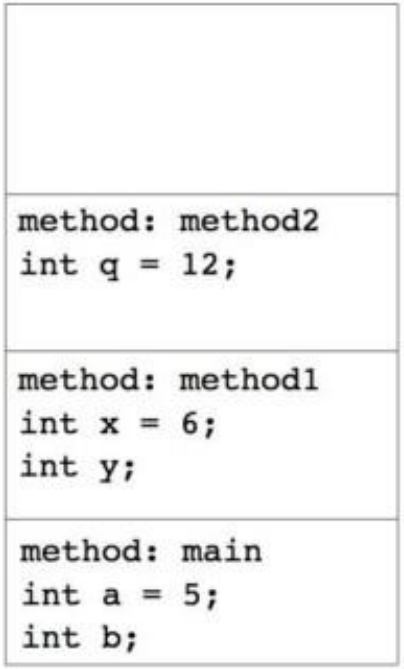
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

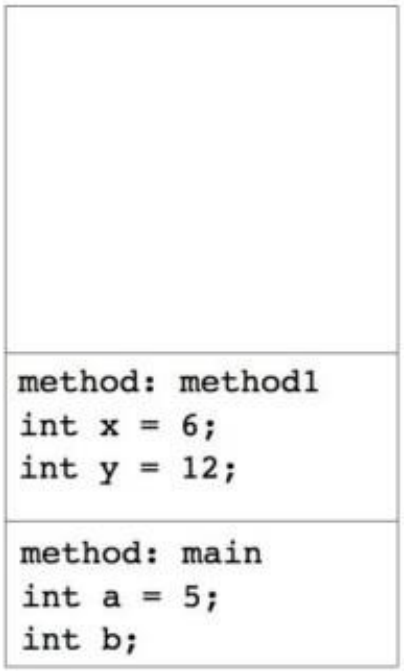
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

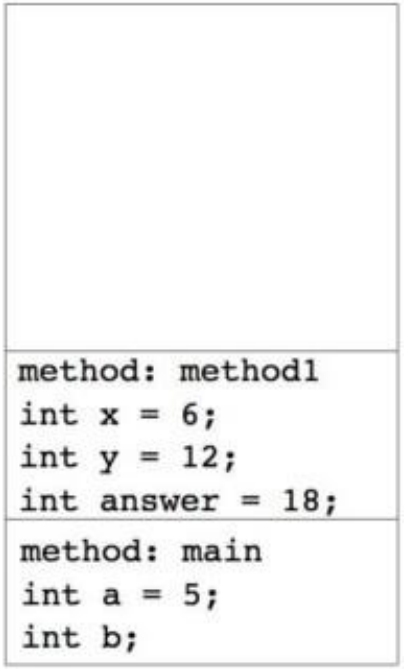
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

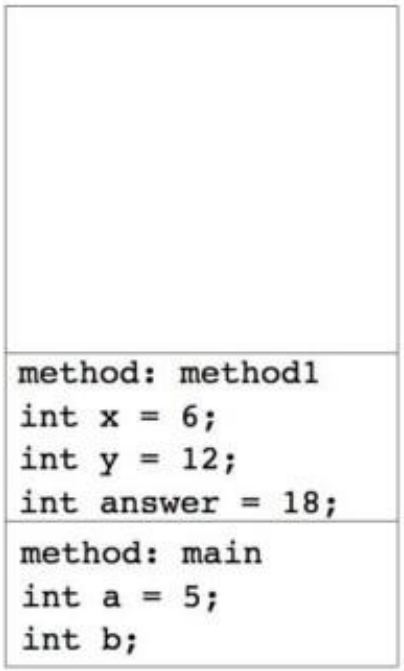
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

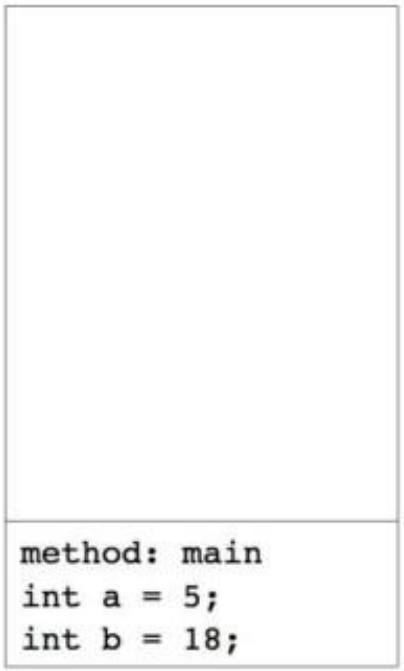
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The Stack

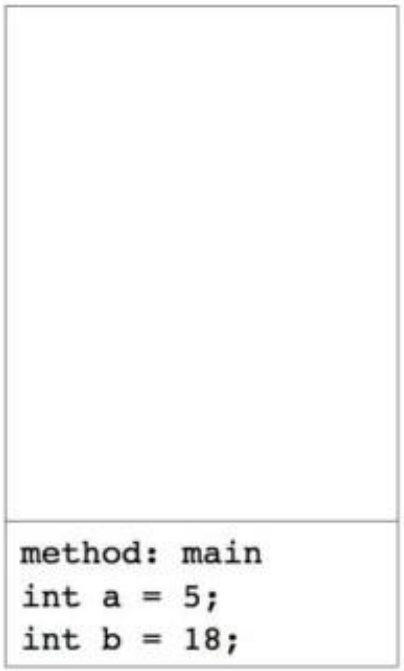
Source Code

```
public static void main(String[] args)
{
    int a = 5;
    int b = method1(a);
    System.out.println(b);
}

public static int method1(int x)
{
    x += 1;
    int y = method2(x);
    int answer = x + y;
    return answer;
}

public static int method2(int q)
{
    q *= 2;
    return q;
}
```

The Stack



The “switch” selection structure

The "switch" statement

- There are times when you are programming that you need to evaluate multiple conditions in succession. For example:

```
int status = 2;

if (status == 1)
{
     System.out.println("Status 1 code ..");
}
else if (status == 2)
{
     System.out.println("Status 2 code ..");
}
else
{
     System.out.println("Invalid status!");
}
```

The "switch" statement

- Java provides a "switch" statement to simplify these kinds of tasks. A switch statement executes statements based on the value of a variable or an expression.

- For example:

```
int status = 2;

switch (status)
{
      case 1: System.out.println("Status == 1"); break;
      case 2: System.out.println("Status == 2"); break;
      default: System.out.println("Invalid!"); break;
}
```

The "switch" statement

- You never "need" to use a switch statement. You can always write a program using "if", "else" and "else if" statements instead of using "switch".
- Switch statements always begin with the "switch 'keyword and a variable enclosed in a set of parenthesis. This variable is the one that is being evaluated in the switch statement.
- Next comes a block of execution.

```
switch (status)

{

   …

}
```

The "switch" statement

- Next comes a series of "case" statements.
- The case statements will be evaluated in order until a match can be found with the variable identified in the switch statement above. Case statements are terminated with a colon.
- Important Note! Case statements do not have blocks of execution associated with them.
- The break command inside of a case statement tells the "switch" statement to end.
- The "default" case is like the "else" block of an "if" statement. It executes if all of the blocks above evaluate to false.

```
switch (status)
{


   case 1: System.out.println("case 1"); break;
   case 2: System.out.println("case 2"); break;

   default: System.out.println("default"); break;



}
```

The "switch" statement

- Note that the "break" command is optional in a switch statement, but you almost always want to use it.
- If it you do not, the switch statement will continue and will treat all future cases as "matches".

Programming Challenge

- Write a program that asks the user to enter in a class year (1,2,3 or 4)
- Determine if the user is a Freshman, Sophomore, Junior or Senior based on this information. Use a switch statement.
- ***ClassYearSwitch.java***

Programming Challenge

- The Chinese Zodiac is based on a twelve-year calendar cycle, with each year represented by an animal

- Write a program to find out the Chinese Zodiac for a given year using a switch statement.

- ***ChineseZodiac.java***