Lecture 4

Methods

NYU

Problem

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

NYU

Problem

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

Solution

```
public static int sum(int i1, int i2) {
   int sum = 0;
   for (int i = i1; i <= i2; i++)
      sum += i;
   return sum;
}
```

```
public static void main(String[] args) {
   System.out.println("Sum from 1 to 10 is " +
sum(1, 10));
   System.out.println("Sum from 20 to 30 is " +
sum(20, 30));
   System.out.println("Sum from 35 to 45 is " +
sum(35, 45));
}
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.
- It is equivalent to a function in python (`def`)

Define a method

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Invoke a method

```
 int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Method header

```
public static int max(int num1, int num2) {

  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

```
int z = max(x, y);
```

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Modifier

```
public static int max(int num1, int num2) {

  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

```
  int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Return value type

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Method name

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Parameter list

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Formal parameters

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Method signature

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
  int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Method body

```
public static int max(int num1, int num2) {

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

```
int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Return value

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

NYU

Defining Methods

- A method is a collection of statements that are grouped together to perform an operation.

Actual parameter (arguments)

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
  int z = max(x, y);
```

NYU

Method Signature

- ***Method signature*** is the combination of the method name and the parameter list.

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

NYU

Formal Parameters

- The variables defined in the method header are known as ***formal parameters***.

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

Actual Parameters

- When a method is invoked, you pass a value to the parameter. This value is referred to as ***actual parameter*** or ***argument***.

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
  int z = max(x, y);
```

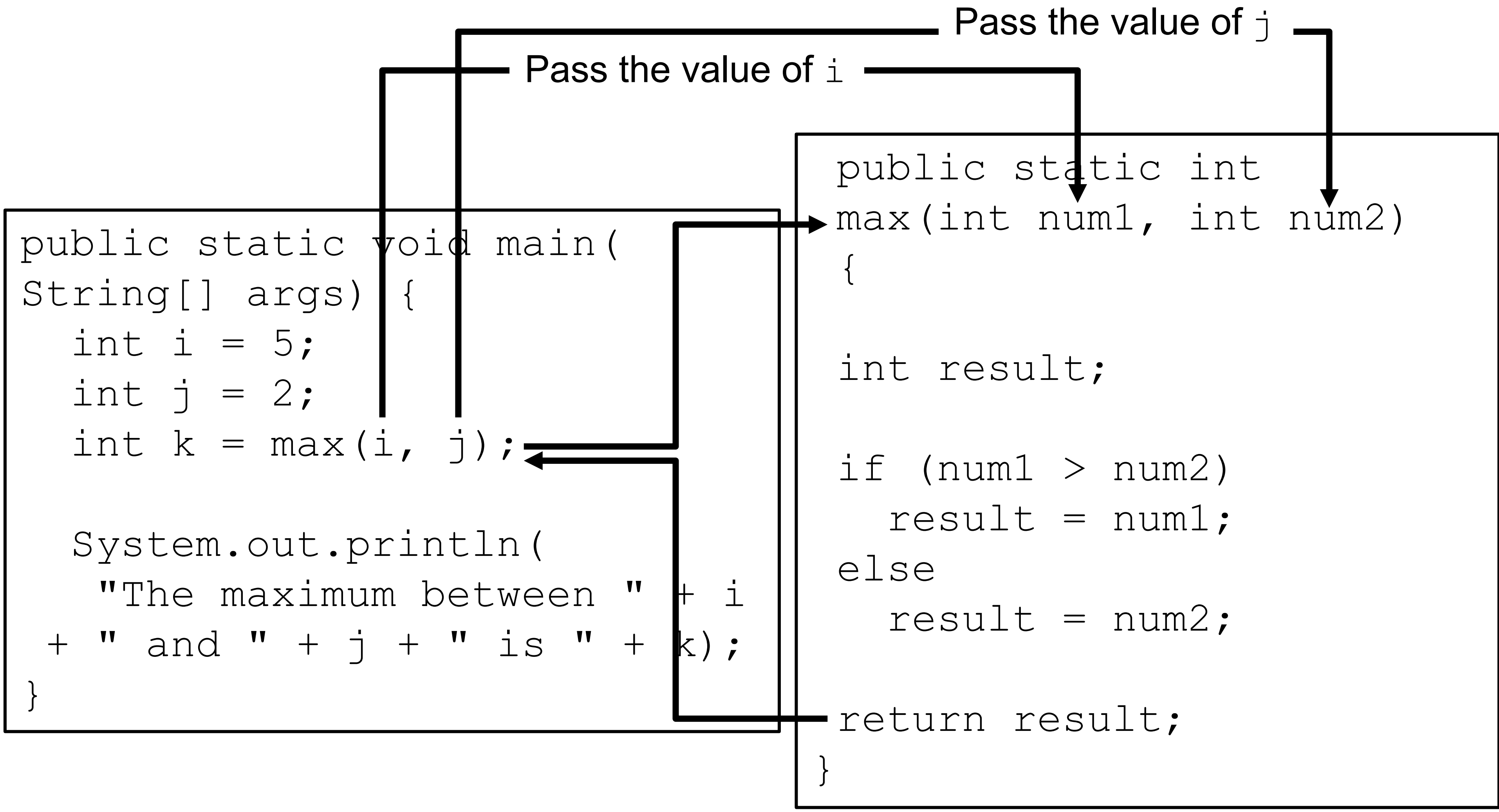
Return Value Type

- The ***returnValueType*** is the data type of the value the method returns. If the method does not return a value, the ***returnValueType*** is the keyword ***void***.

```
 public static int max(int num1, int num2) {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

```
 int z = max(x, y);
```

Calling Methods



```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

NYU

Trace Method Invocation

`i` is now 5

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

Trace Method Invocation

`j` is now 2

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

NYU

Trace Method Invocation

invoke `max(i, j)`

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

NYU

Trace Method Invocation

invoke `max(i, j)`
Pass the value of `i` to `num1`
Pass the value of `j` to `num2`

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
 {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

NYU

Trace Method Invocation

declare variable result

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int
 max(int num1, int num2)
 {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Trace Method Invocation

(num1 > num2) is true
since num1=5 and num2= 2

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Trace Method Invocation

result is now 5

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

NYU

Trace Method Invocation

return `result`, which is 5

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int
 max(int num1, int num2)
 {

 int result;

 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

NYU

Trace Method Invocation

return `max(i, j)` and assign the return value to `k`

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

NYU

Trace Method Invocation

Execute the print statement

```
public static void main(
String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
public static int
max(int num1, int num2)
{

int result;

if (num1 > num2)
  result = num1;
else
  result = num2;

return result;
}
```

NYU

Caution

- A return statement is **required** for a value-returning method.
- Example:

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

$$\longrightarrow$$

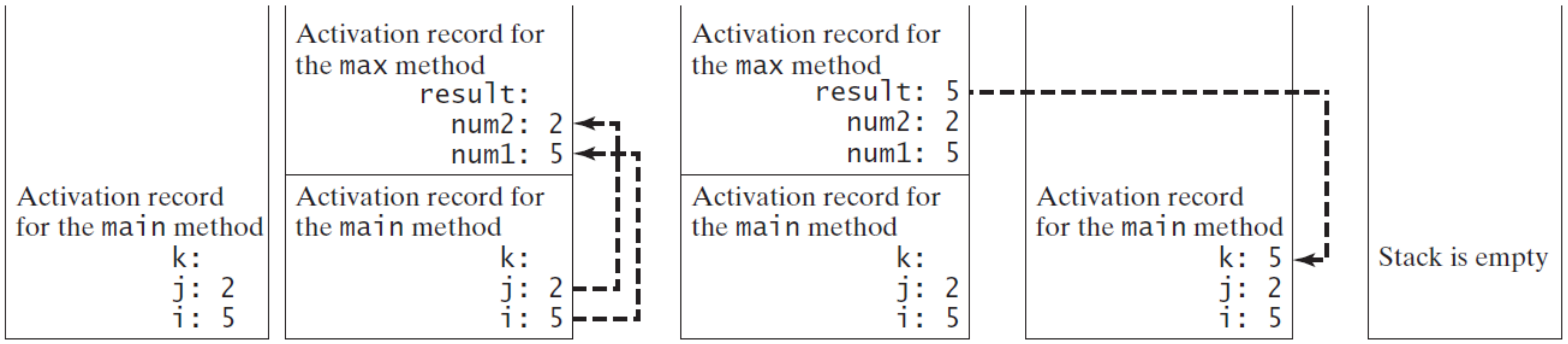
```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

- To fix this problem:
  - delete `if (n < 0)`, so that the compiler will see a `return` statement to be reached regardless of how the `if` statement is evaluated

Reuse Methods from Other Classes

- One of the benefits of methods is for reuse.
- The max method can be invoked from any class besides `TestMax`.
- If you create a new class `Test`, you can invoke the max method using `ClassName.methodName` (e.g., `TestMax.max`).

Call Stacks

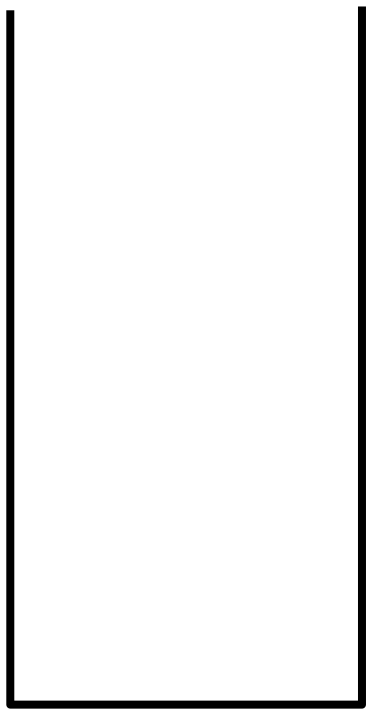


Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result:
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5
Activation record for the max method
result: 5
num2: 2
num1: 5
Activation record for the main method
k:
j: 2
i: 5
Activation record for the main method
k: 5
j: 2
i: 5
Stack is empty

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```



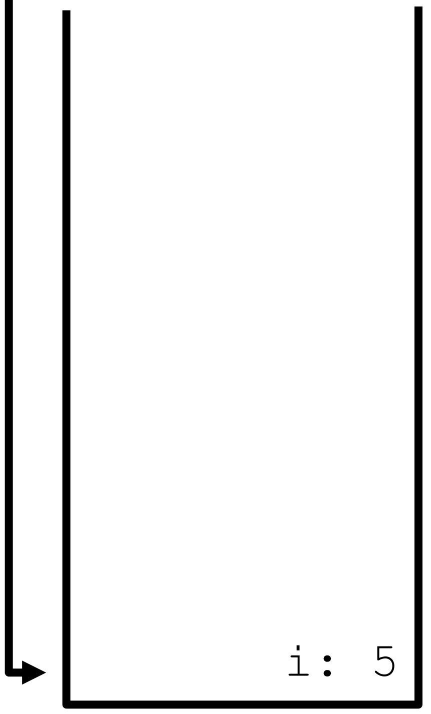
The main method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

i is declared and initialized



The main method is invoked

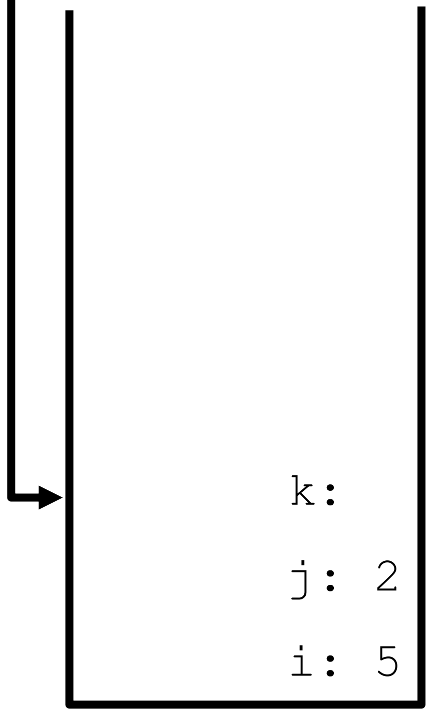
NYU

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

k is declared



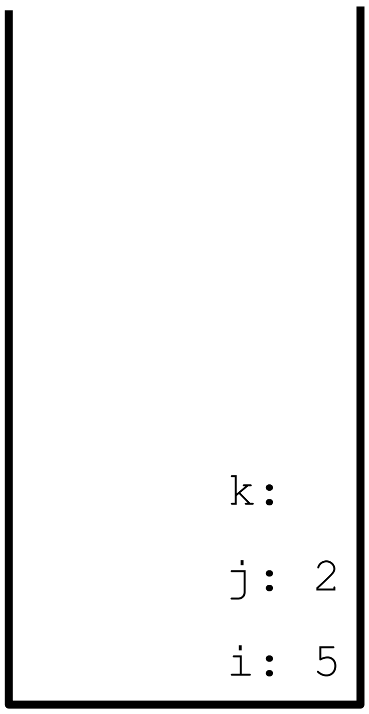
The main method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

Invoke `max(i, j)`

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```



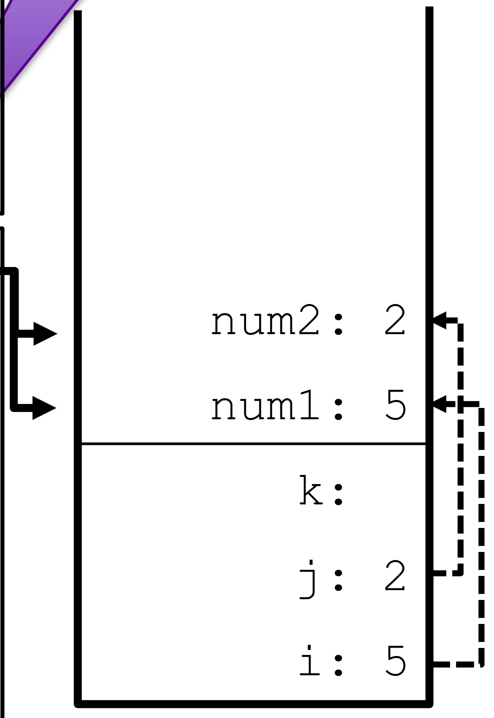
The main method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

pass the values of `i` and `j` to `num1` and `num2`



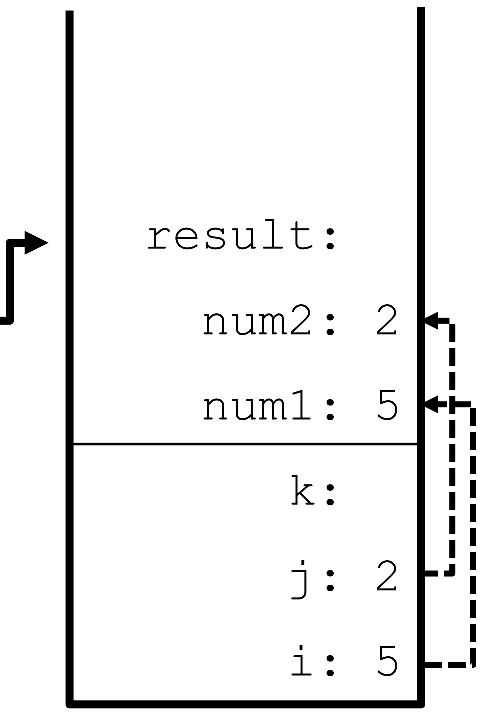
The max method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Declare `result`



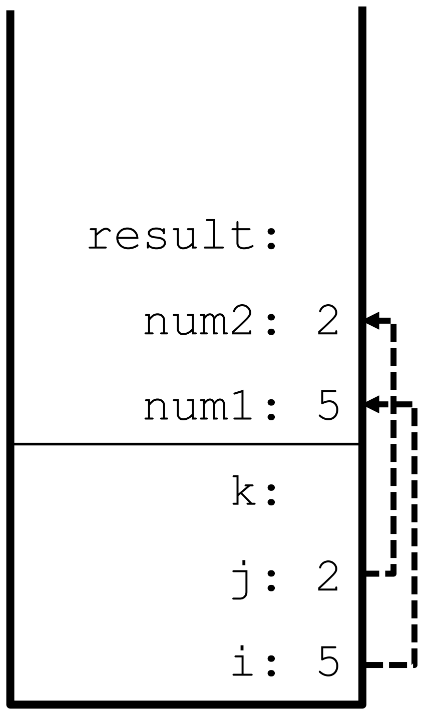
The max method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

`(num1 > num2)` is true



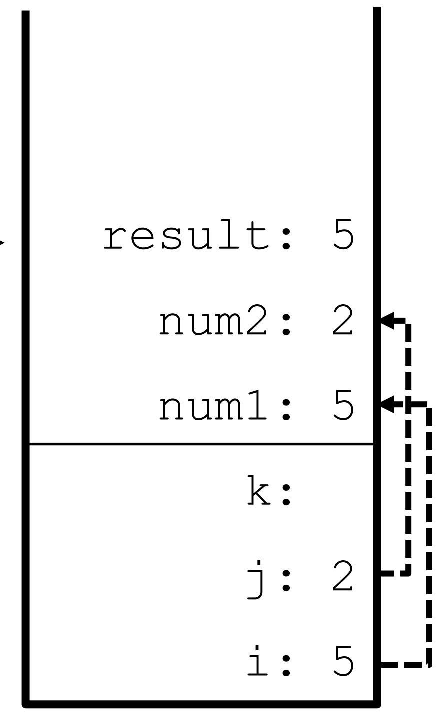
The max method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Assign num1 to result



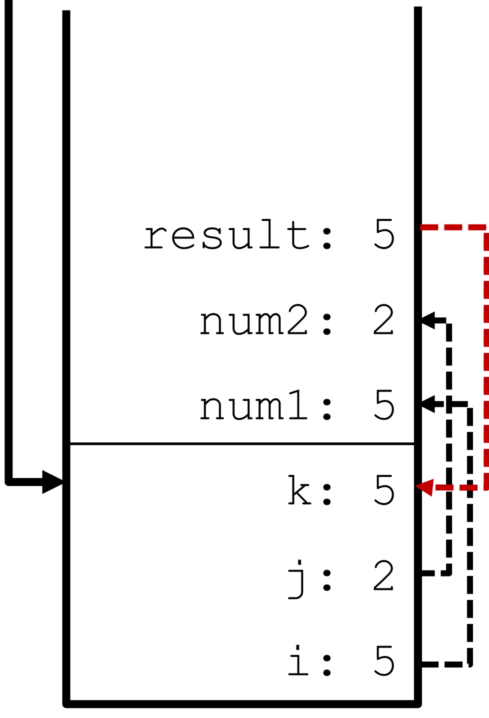
The max method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```

Return `result` and assign it to `k`



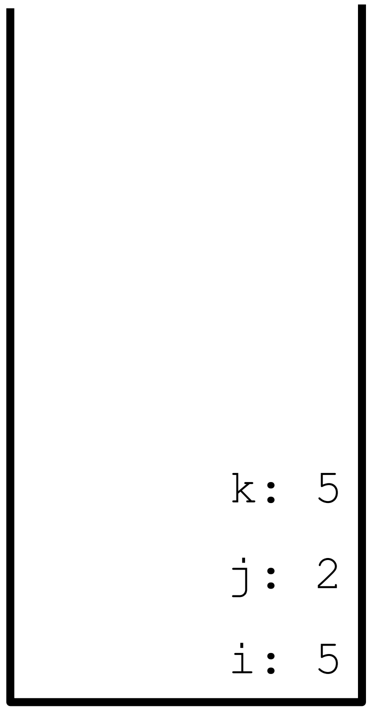
The max method is invoked

Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i
 + " and " + j + " is " + k);
}
```

Execute print statement

```
 public static int max(int num1, int num2) {
 int result;
 if (num1 > num2)
   result = num1;
 else
   result = num2;

 return result;
}
```



The main method is invoked

void Method Example

- Use `void` for methods that do not return values
- Example:
  - `public static` **`void`** `main(String[] args)`
- This type of method does not return a value.
- The method performs some actions.

NYU

Passing Parameters

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

- Suppose you invoke the method using
  - nPrintln(“Welcome to Java”, 5);
- What is the output?

- Suppose you invoke the method using
  - nPrintln(“Computer Science”, 15);
- What is the output?

- Can you invoke the method using
  - nPrintln(15, “Computer Science”);

Pass by Value

```
public class Increment {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the call, x is " + x);
    increment(x);
    System.out.println("After the call, x is " + x);
  }
  public static void increment(int n) {
    n++;
    System.out.println("n inside the method is " + n);
  }
}
```

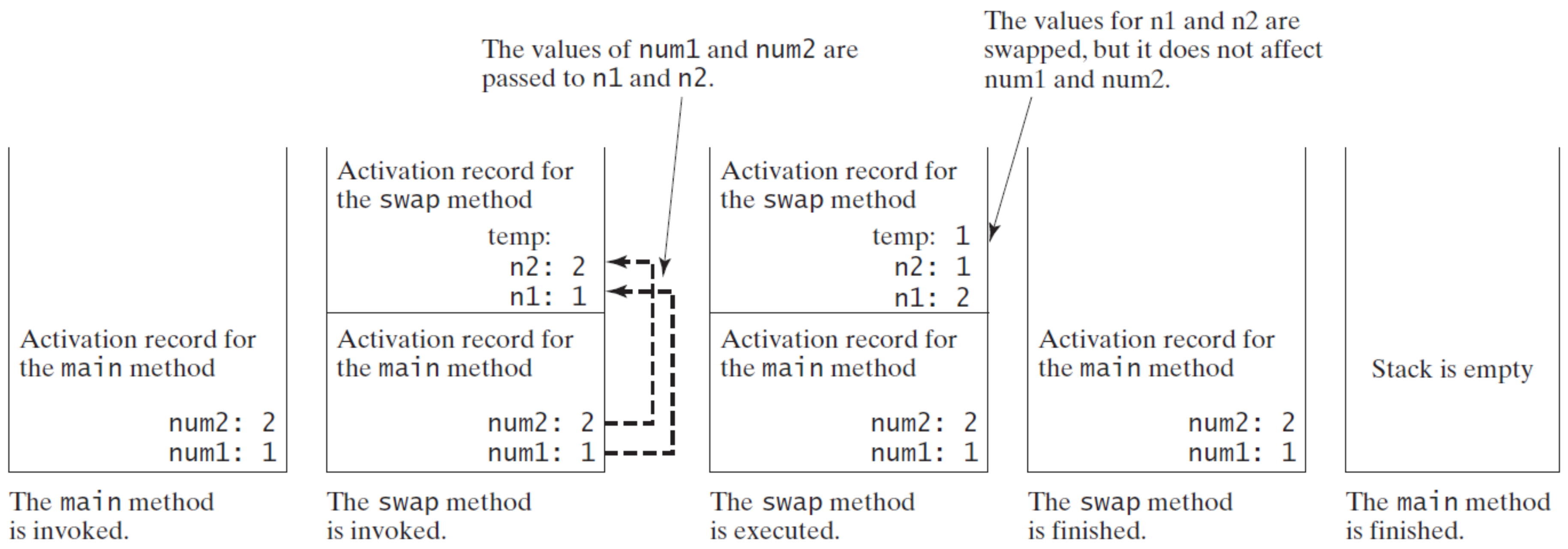
NYU

Pass by Value

```
public class TestPassByValue {
 public static void main(String[] args) {
   int num1 = 1;
   int num2 = 2;
   System.out.println("Before, num1 is " + num1 + " and num2 is " + num2);
   swap(num1, num2);
   System.out.println(”After, num1 is " + num1 + " and num2 is " + num2);
  }

 public static void swap(int n1, int n2) {
  System.out.println("\tInside the swap method");
  System.out.println("\t\tBefore, n1 is " + n1      + " and n2 is " + n2);
  // Swap n1 with n2
  int temp = n1;
  n1 = n2;
  n2 = temp;
  System.out.println("\t\tAfter, n1 is " + n1      + " and n2 is " + n2);
  }
}
```

NYU

Pass by Value



The values of num1 and num2 are passed to n1 and n2.
The values for n1 and n2 are swapped, but it does not affect num1 and num2.
Activation record for the main method
num2: 2
num1: 1
The main method is invoked.
Activation record for the swap method
temp:
n2: 2
n1: 1
Activation record for the main method
num2: 2
num1: 1
The swap method is invoked.
Activation record for the swap method
temp: 1
n2: 1
n1: 2
Activation record for the main method
num2: 2
num1: 1
The swap method is executed.
Activation record for the main method
num2: 2
num1: 1
The swap method is finished.
Stack is empty
The main method is finished.

Modularizing Code

- reduce redundant coding.
- enable code reuse.
- modularize code
- improve the quality of the program.

Overloading Methods

- Overloading the max Method

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

NYU

Ambiguous Invocation

- Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
- This is referred to as ambiguous invocation.
- Ambiguous invocation is a compile error.

NYU

Ambiguous Invocation

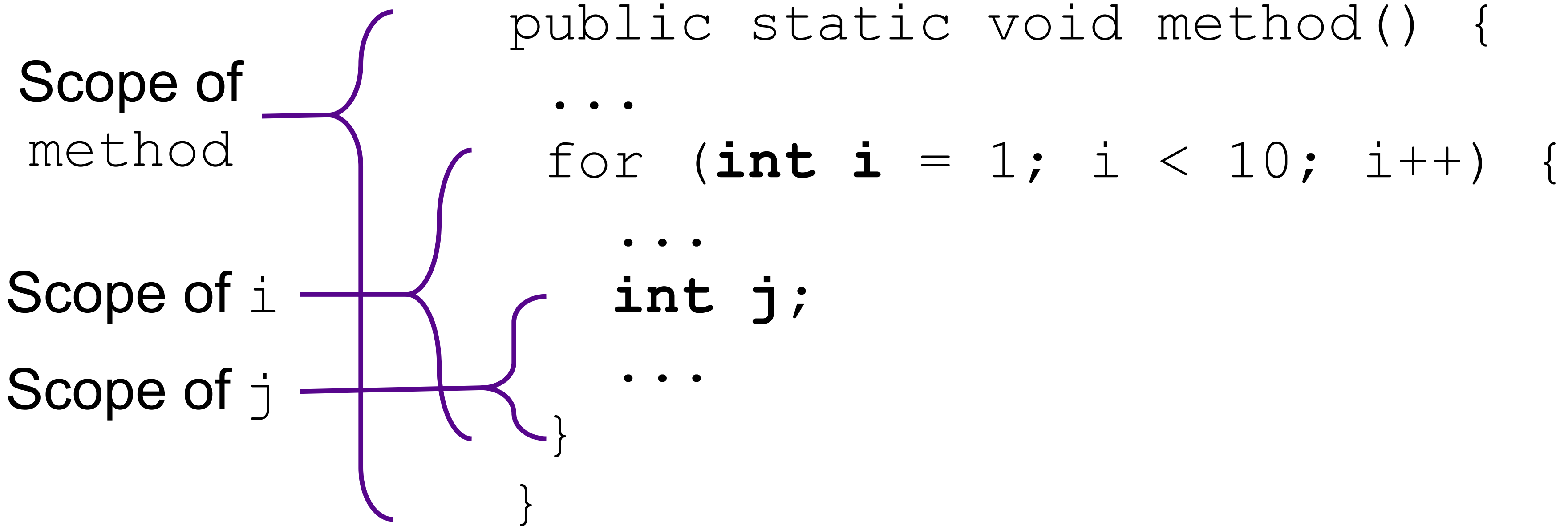
```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```

Scope of Local Variables

- A local variable: a variable defined inside a method.
- Scope: the part of the program where the variable can be referenced.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
- A local variable must be declared before it can be used.
- You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.

Scope of Local Variables

- A variable declared in the initial action part of a for loop header has its scope in the entire loop. But a variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.



```
public static void method() {
  ...
  for (int i = 1; i < 10; i++) {
    ...
    int j;
    ...
  }
}
```

NYU

Scope of Local Variables

```
public static void method1() {
   int x = 1;
   int y = 1;

   for (int i = 1; i < 10; i++) {
      x += i;
   }

   for (int i = 1; i < 10; i++) {
      y += i;
   }
}
```

NYU

Scope of Local Variables

```
public static void method2() {

    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++) {
       sum += i;
    }

 }
```

NYU

Scope of Local Variables

```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

NYU

Scope of Local Variables

```
// Fine with no errors
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i is declared
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i is declared again
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

NYU

Scope of Local Variables

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

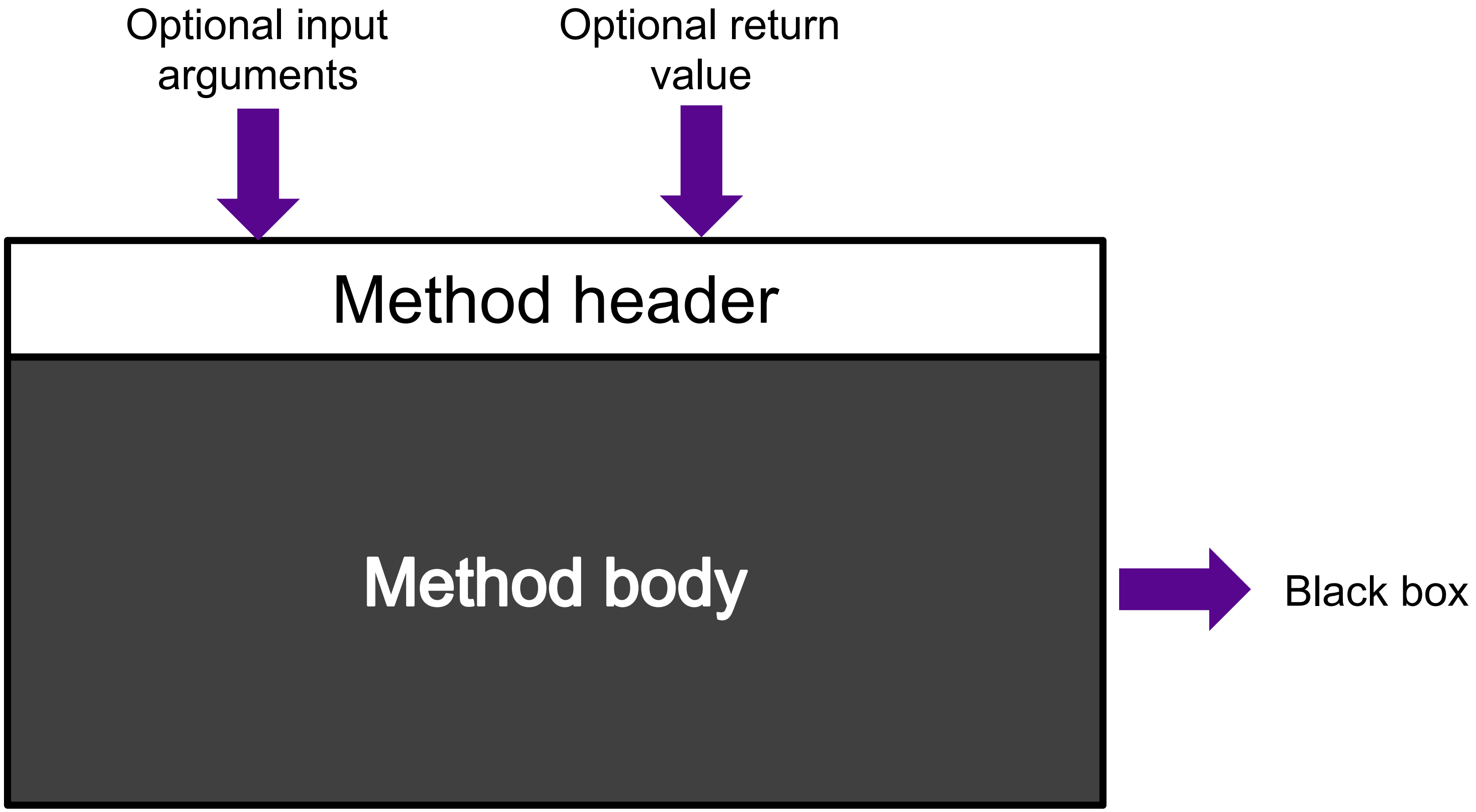
Scope of Local Variables

```
// With errors
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

NYU

Method Abstraction

- You can think of the method body as a black box that contains the detailed implementation for the method.



NYU

Benefits of Methods

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

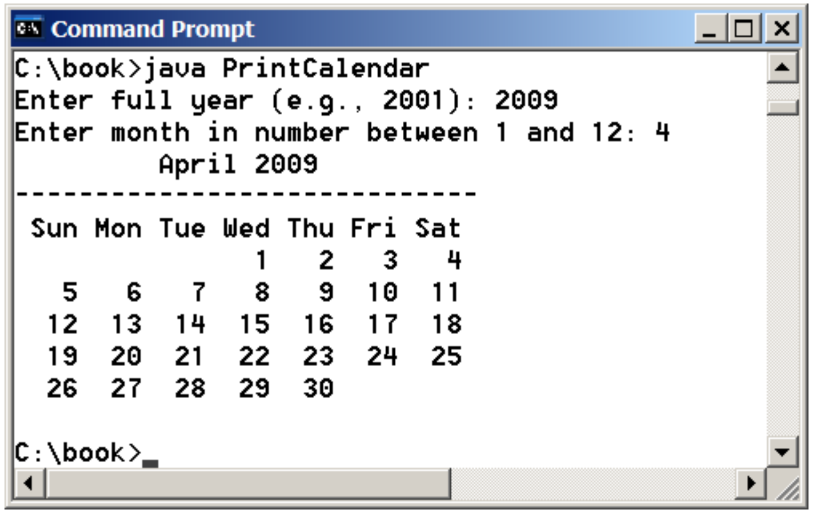
Stepwise Refinement

- The concept of method abstraction can be applied to the process of developing programs.
- When writing a large program, you can use the “divide and conquer” strategy, also known as stepwise refinement, to decompose it into subproblems.
- The subproblems can be further decomposed into smaller, more manageable problems.

NYU

PrintCalender Case Study

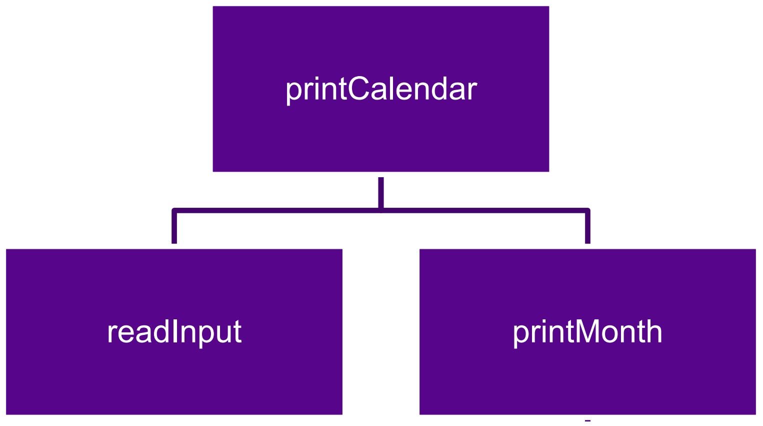
- Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.



```
C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
         April 2009
-----------------------------
 Sun Mon Tue Wed Thu Fri Sat
                   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

C:\book>_
```

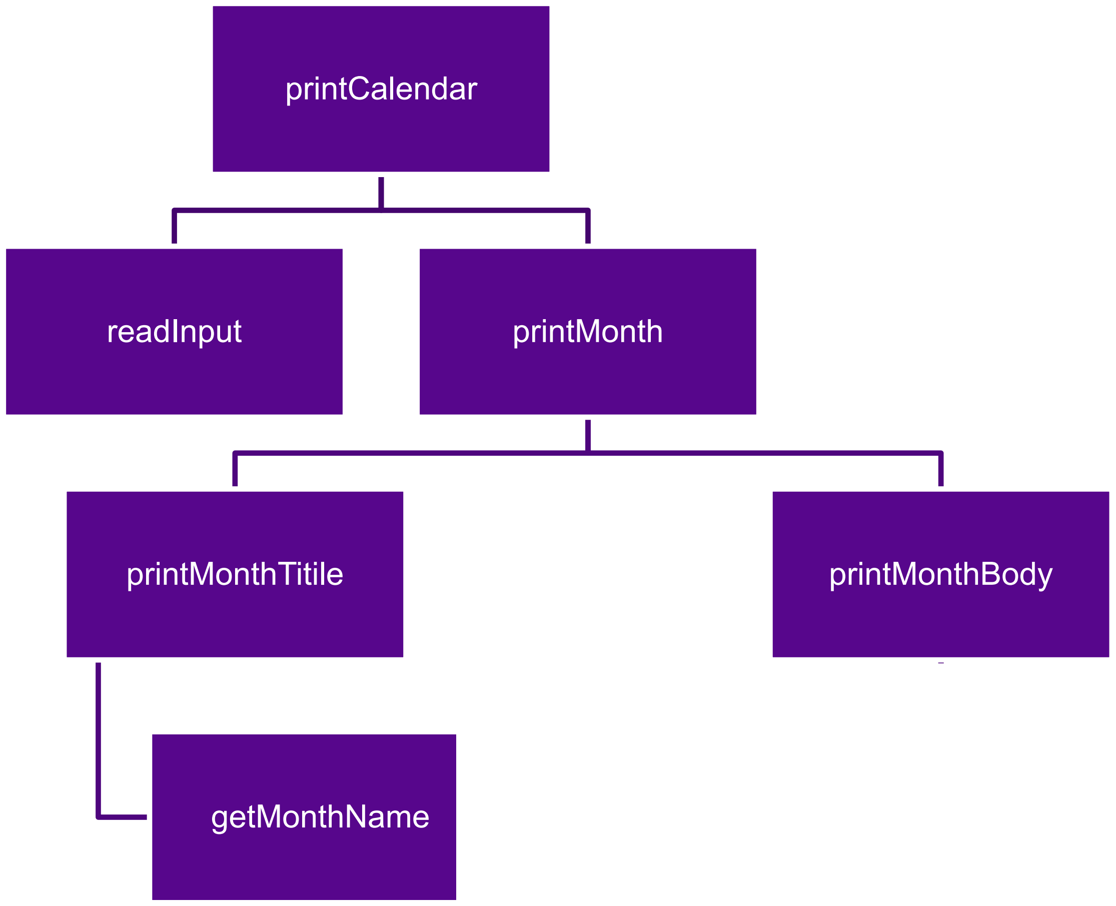
NYU

Design Diagram

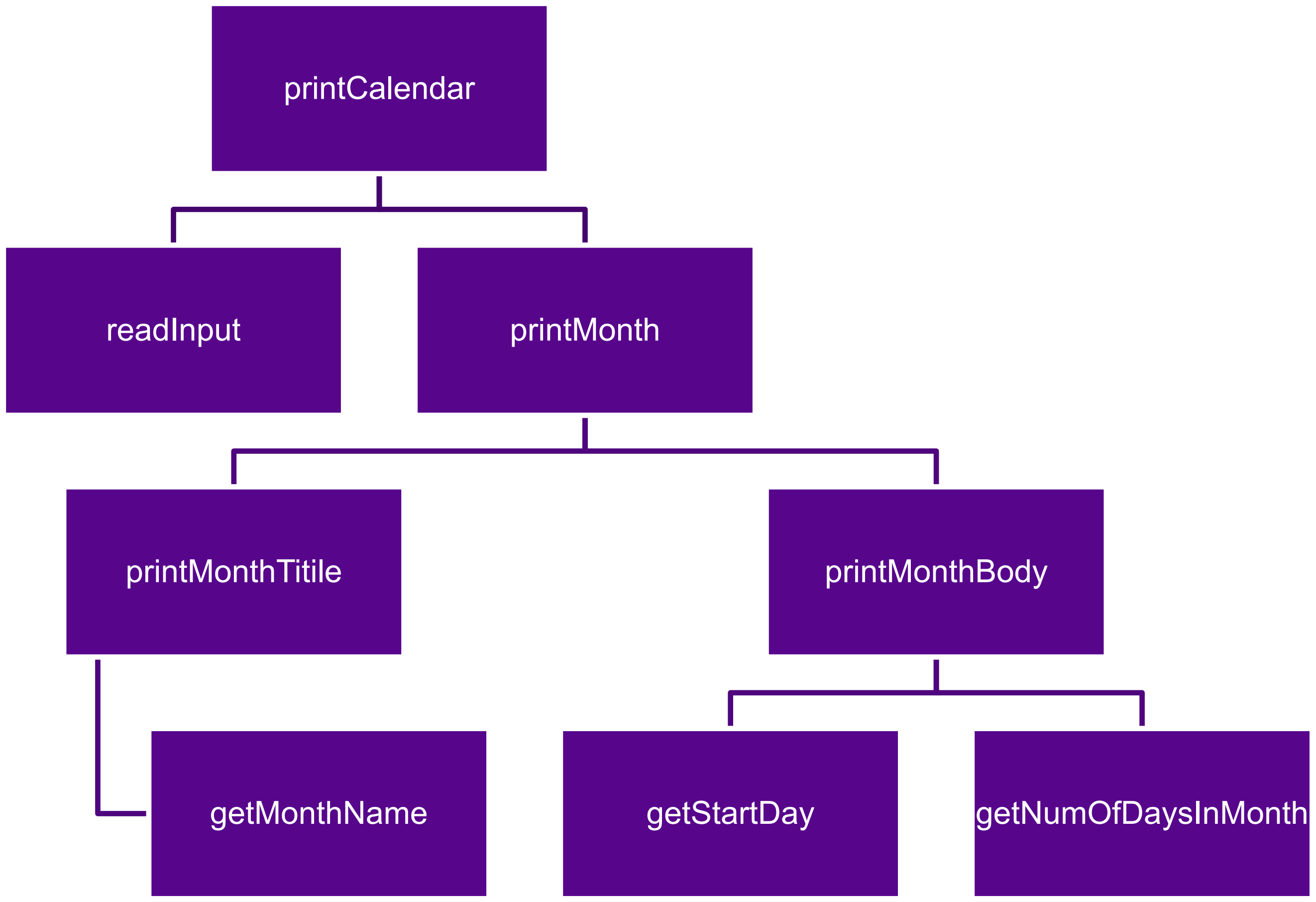


NYU

Design Diagram

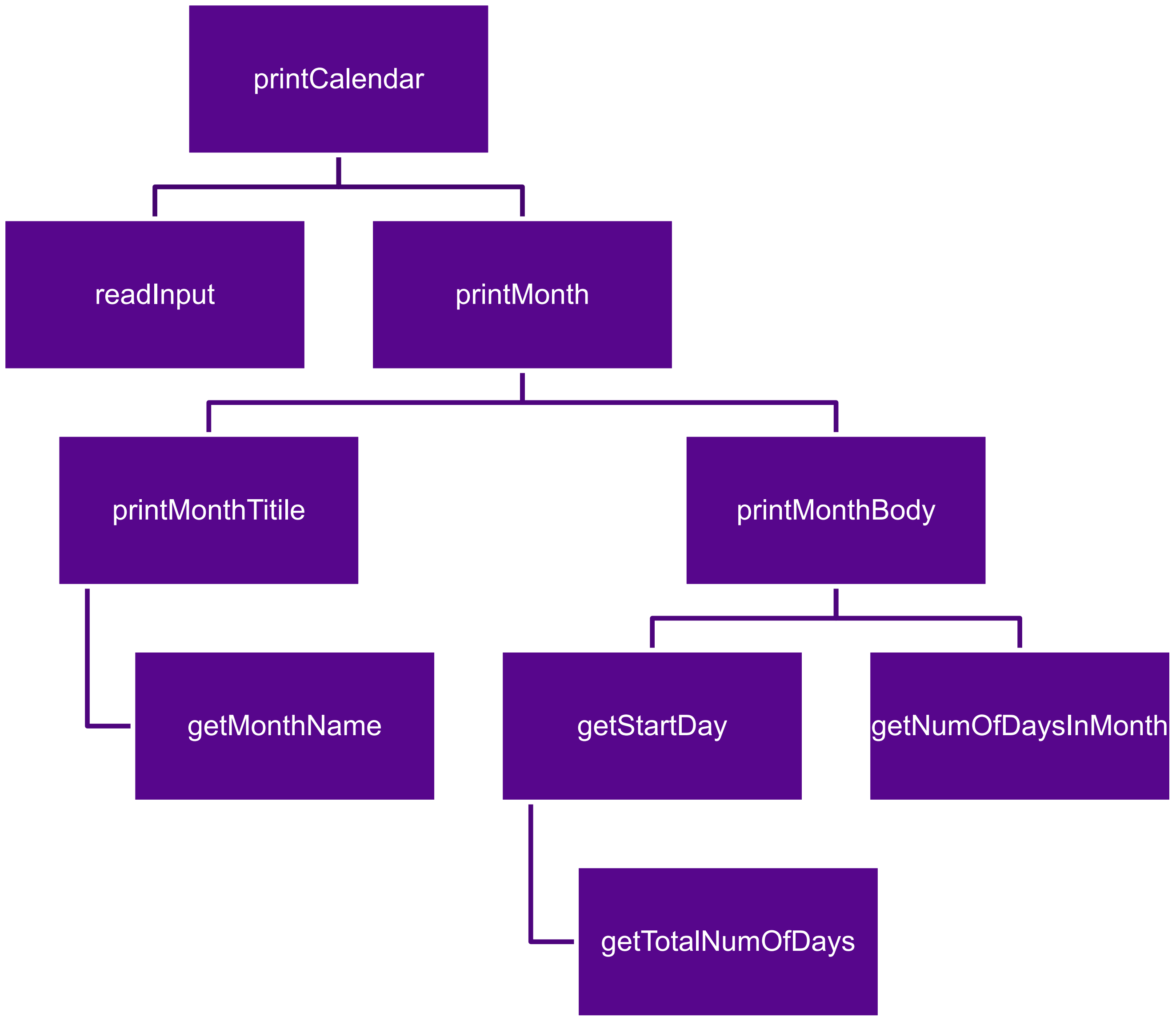


Design Diagram



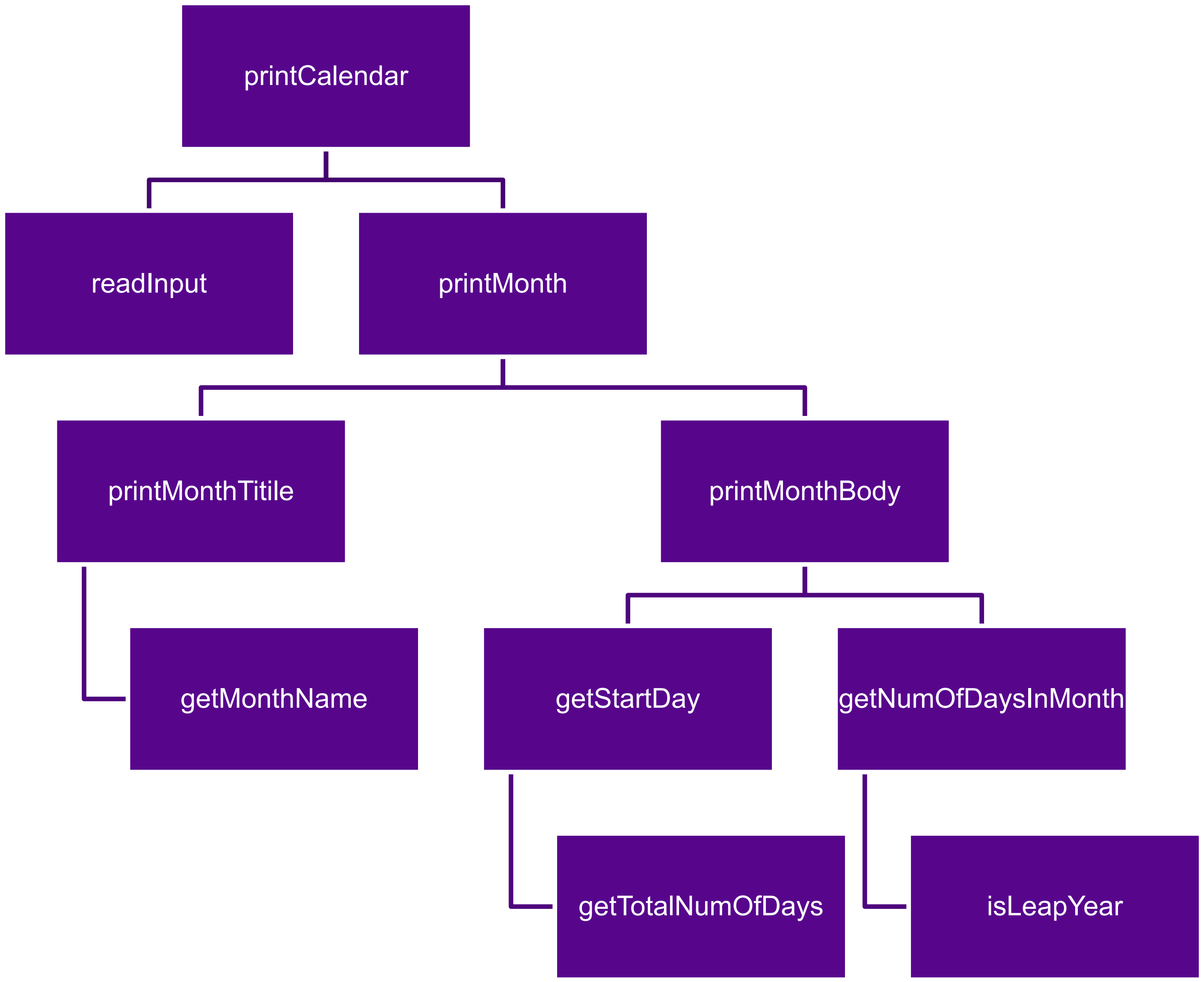
NYU

Design Diagram



NYU

Design Diagram



NYU

Implementation: Top-Down

- Top-down approach is to implement one method in the structure chart at a time from the top to the bottom.
- Stubs can be used for the methods waiting to be implemented.
- A stub is a simple but incomplete version of a method.
- The use of stubs enables you to test invoking the method from a caller.
- Implement the main method first and then use a stub for the printMonth method.

NYU

Implementation: Bottom-Up

- Bottom-up approach is to implement one method in the structure chart at a time from the bottom to the top.
- For each method implemented, write a test program to test it.
- Both top-down and bottom-up methods are fine. Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy.
- Sometimes, they can be used together.

NYU

Benefits of Stepwise Refinement

- Simpler Program
- Reusing Methods
- Easier Developing, Debugging, and Testing
- Better Facilitating Teamwork