```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

This makes a 5 row array that has the values 0, 1, 3, 6, 10 (i+values[i-1]), then makes the first value 11 (value [1] + value[4])

```
Scanner keyboard = new Scanner(System.in);

int size = keyboard.nextInt();
double[] numbers = new double[size];
for (int i=0; i<numbers.length; i++){
     numbers[i]=keyboard.nextDouble();
}

keyboard.close();
```

This creates an array of size (next integer), then initilaizes with double values in a loop with “size” number of values

Elements in array don’t have to be primitive types
A primitive type is a basic type, such as int double char boolean. It’s a single value, and stored directly in memory. They are not objects.
A reference type refers to an object, such as

```
String s1 = "hello";
String s2 = s1;
s2 = "world";
System.out.println(s1); // Output: hello
```

Here s2 is assigned reference to s1, and it doesnt change s1 when s2 turns into “world”
The scope of an array in Java begins when the array is declared and ends when the array goes out of scope. The scope of an array determines the visibility and lifetime of the array.

An array's scope begins when it is declared, and it ends when the code block in which it is declared is exited. If an array is declared within a method, its scope begins when the method is called and ends when the method returns. If an array is declared within a class, its scope begins when an instance of the class is created and ends when the instance is garbage collected.

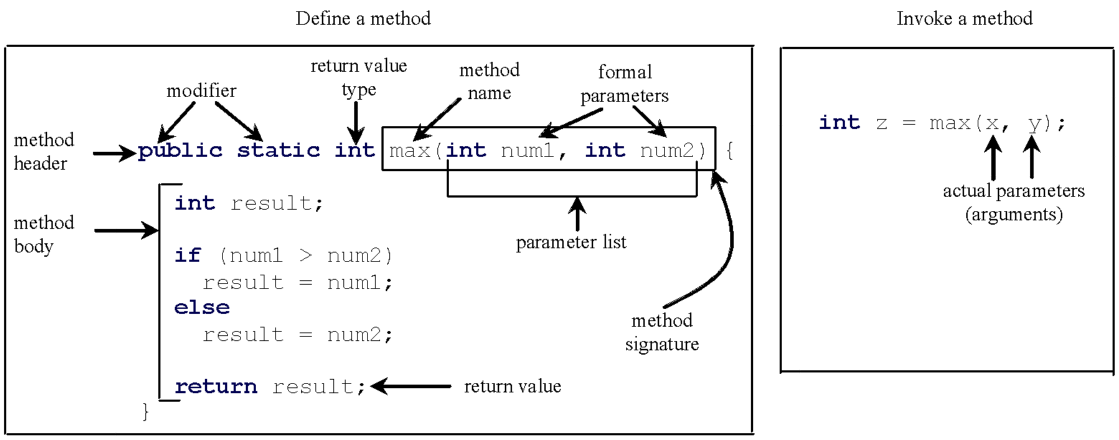
```java
String[] womenInTech = {"Sandberg, Sheryl", "Schneider, Erna", "Bartik, Jean",
"Klawe, Maria", "Hopper, Grace", "Borg, Anita", "Lovelace, Ada"};

String key = "Lovelace, Ada";
System.out.println(key.compareTo(womenInTech[0]));
System.out.println(key.compareTo(womenInTech[1]));
System.out.println(key.compareTo(womenInTech[2]));
System.out.println(key.compareTo(womenInTech[3]));
System.out.println(key.compareTo(womenInTech[4]));
System.out.println(key.compareTo(womenInTech[5]));
System.out.println(key.compareTo(womenInTech[6]));
```

The method compareTo() compares strings lexicographically (compares unicode values in sequence)

If the string is equal, it returns

Methods

Method is collection of statements that are grouped together to perform an operation

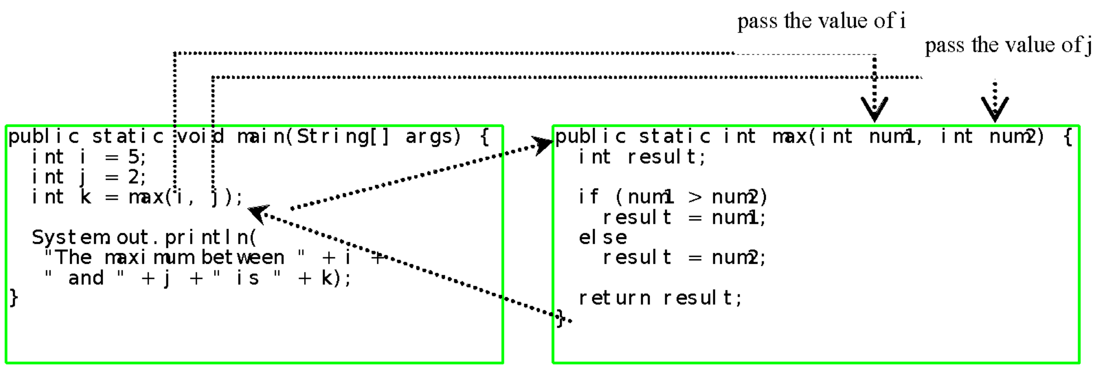


```java
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

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

Int result is declaring the variable result, return value

Scope of the variables are within the method, such as



```java
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);
}
```

```java
public static int max(int num1, int num2) {
  int result;

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

  return result;
}
```

Here scope of “i j k” are in main, and num1 num2 are in max
Methods are flexible code containers
You can create other methods in same class but not the main void method
Needs predefined input and output
We call it “invoking” a method

Example : CardMethods

```java
public class CardsMethods {
   public static void main(String[] args) {
       String suitName = "", cardIdentity = "";
       int card = generateCard();
       String suit = findSuit(card);
       String identity = findIdentity(card);
       // Display card identity and suit name
       System.out.println("You picked the " + identity + " of " + suit);
   }
      public static String findSuit(int cardNum){
       String suitName;
       int suit = cardNum / 13;
       // Finding the suit name
       if (suit == 0) // 0-12
           suitName = "Diamonds";
       else if (suit == 1) // 13-25
           suitName = "Clubs";
       else if (suit == 2)
           suitName = "Hearts";
       else
           suitName = "Spades";
       return suitName;
   }
      public static int generateCard(){
  int num = (int) (Math.random()*52);
  return num;
 }
public static String findIdentity(int cardNum){
   int identity = (cardNum+1)%13;
   String cardIdentity = "";
   // Finding the card identity
   switch(identity) {
       case 0: cardIdentity = "King"; break;
       case 1: cardIdentity = "Ace"; break;
       case 2: cardIdentity = "2"; break;
       case 3: cardIdentity = "3"; break;
       case 4: cardIdentity = "4"; break;
       case 5: cardIdentity = "5"; break;
       case 6: cardIdentity = "6"; break;
       case 7: cardIdentity = "7"; break;
       case 8: cardIdentity = "8"; break;
       case 9: cardIdentity = "9"; break;
       case 10: cardIdentity = "10"; break;
       case 11: cardIdentity = "Jack"; break;
       case 12: cardIdentity = "Queen"; break;
   }
   return cardIdentity;
}
```

The Pass By Value / Reference

Passing by value is a copy of current value of arg to a method

PassBy Reference is a pointer (ref variable) to memory location of arg to method

When datat type is primitive, actual value is passed

For array type, reference is used to so changes are synchronized

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

Method for reversing an array

```

public class PassBy{
   public static void main(String[]args){
       int[] numbers = {100, 200, 300, 400};
       printArray(numbers);
       int[] newNumbers = copyArray(numbers);
       doubleArrayContent(numbers);
       printArray(numbers);
       //int[] newNumbers = copyArray(numbers);
       printArray(newNumbers);
   }
   public static void printArray(int[] list){
       for(int i=0; i < list.length; i++){
           System.out.print(list[i]+ " ");
       }
 System.out.println();
      }
   public static void doubleArrayContent(int[] list){
       for (int i=0; i<list.length; i++)
       list[i] = list[i] * 2;
   }
   public static int[] copyArray(int[] list){
       int[] newList = new int[list.length];
       for(int i=0; i<list.length; i++){
       newList[i] = list[i];
       }
       return newList;
      }
}
```

 This will return 100 200 300 400 200 400 600 800 100 200 300 400

Heap Memory:

The heap is a region of memory used for dynamic memory allocation in Java.
The heap is shared by all parts of the program and is used to store objects and other data structures.
The heap is managed by the garbage collector, which automatically frees memory that is no longer used by the program.

Stack:

The stack is a region of memory used for storing method calls and local variables in Java.
The stack is organized in a Last In First Out (LIFO) manner, meaning that the last method call made is the first one to return.
The stack has a limited size, and if too many method calls are made, a StackOverflowError can occur.

Pre/Post Conditions:

Preconditions are conditions that must be satisfied before a method is called.
Postconditions are conditions that must be satisfied after a method is called.
Pre- and post-conditions are used to specify the behavior of a method and to ensure that the method behaves as expected.
Pre- and post-conditions are specified using assertions in Java.