Why Methods?

- Suppose we want the sum of the integers from **1** to **10**, from **20** to **30**, and from **42** to **73**:

```
public class Sums1
{
   public static void main( String[] args )
   {
      int i, sum;
      sum = 0;
      for ( i = 1; i <= 10; i++) {
         sum += i;
      }
      System.out.println("Sum from 1 to 10 is " + sum);

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

      sum = 0;
      for ( i = 42; i <= 73; i += 1) {
         sum += i;
      }
      System.out.println("Sum from 42 to 73 is " + sum);
   } // end of method main
} // end of class Sums1
```

Similar, but not identical, loops.

Writing Methods:

- A *method* is a collection of statements grouped together to perform an operation. It is a function defined within a class.
- Can group the **`for`** statement and the declarations of **`i`** and **`sum`** together to create a *method*:

```
public static int findSum( int start, int end)
{
   int i;
   int sum;

   sum = 0;
   for ( i = start; i <= end; i++ ) {
      sum += i;
   }

   return sum;
} // end of method findSum
```

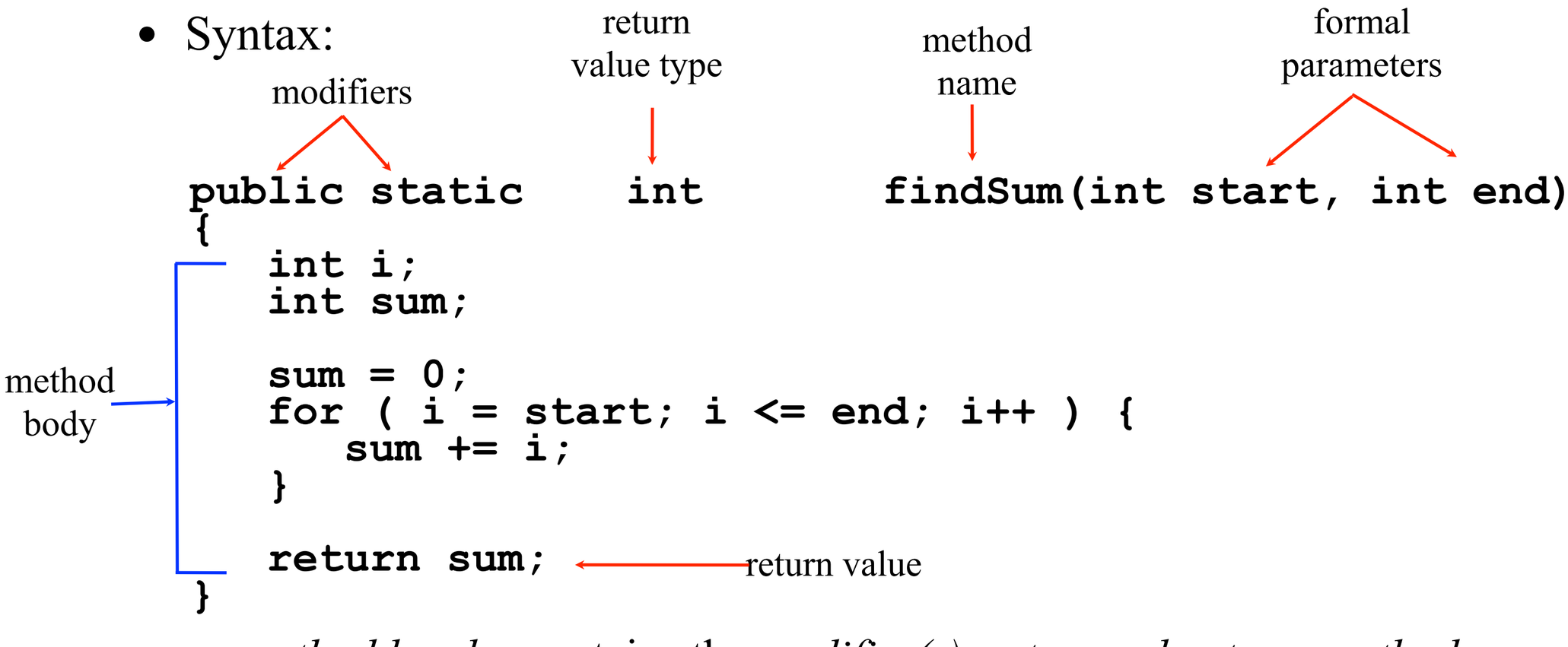
- Can then *invoke* this *method* each time we want to compute a new sum.

```
public class Sums2
{
   public static int findSum( int start, int end)
   {
      int i;
      int sum;

      sum = 0;
      for ( i = start; i <= end; i++ ) {
         sum += i;
      }

      return sum;
   } // end of method findSum

   public static void main( String[] args )
   {

      System.out.println("Sum from 1 to 10 is " + findSum(1, 10) );

      System.out.println("Sum from 20 to 30 is " + findSum(20, 30) );

      System.out.println("Sum from 42 to 73 is " + findSum(42, 73) );
   } // end of method main

} // end of class Sums2
```

The method **findSum**

Can invoke **findSum** from multiple places

- Syntax:



- *method header* contains the *modifier(s)*, *return value type*, *method name*, and *parameters*.
- *method body* contains the code to implement the function. Can use any the Java statements.
- *return value type* is what the method produces. Can be any type or class we have covered (**`int`**, **`long`**, **`String`**, **`double`**, etc.).
  - Can also be **`void`** if the method does not return anything.

Invoking a Method:

- If the method returns a value, a call to the method is usually treated as a value:

```
int answer;
answer = findSum(5, 15);
answer = 7 * findSum(7, 72);

answer = findSum(16, 38) + 8 * findSum(37, 42);
```

  - The method can be invoked anywhere that an identifier of the same type as the *return value type* can appear:

```
System.out.println("The result is " + findSum(3, 13) );
if ( findSum(8, 12) <= xray )
```

Invoking a Method:

- The *arguments* can be identifiers:

```
int begin, finish;
// Ask the user for the starting and ending points of the sum
begin = inputScan.nextInt();

finish = inputScan.nextInt();
// Invoke findSum

answer = findSum( begin, finish);
```

- Can use any type that is compatible with the *formal parameters*:

```
public static int findSum(int start, int end)
```

Invoking a Method (continued):

- The values of the arguments are copied into the formal parameters.
  - The value of **`xray`** is copied to **`start`**. The value of **`yoke`** is copied to **`end`**.
- The method is then executed.
- The return value is given back to **`main`** and **`main`** continues executing.

```
public static void main( String[]
args )
{
   int answer;
   int xray = 93;
   int yoke = 104;
   answer = 17 * findSum( xray,
yoke);
   System.out.println("answer is " +
answer);
} // end of method main
```

```
public static int findSum( int start, int end)
{
   int i;
   int sum;

   sum = 0;
   for ( i = start; i <= end; i++ ) {
      sum += i;
   }

   return sum;
} // end of method findSum
```

Invoking a Method:

- All parameters in Java are what are termed ***call by value***.
  - This means the values are copied, so changing the value of the parameter in the method, will not change the value of the argument in the call. For example

```
int begin, finish;
answer = findSum( begin, finish);
```

  - The values of begin and finish above don't change even if the function does:

```
public static int findSum(int start, int end) {
    start = 1000;
```

Invoking a Method (continued):

- The only thing that comes back from a method is the return value.
  - **xray** and **yoke** are not changed by the call to **findSum**. This is the case even when **findSum** changes the values of **start** and **end**.

```
public class Sums3
{
   public static int findSum( int start, int end)
   {
      int i;
      int sum;

      System.out.print("findSum: start = " + start);
      System.out.println("   end = " + end);

      sum = 0;
      for ( i = start; i <= end; i++ ) {
         sum += i;
      }

      start = 2 * start;
      end = 3 * end;

      System.out.print("findSum: start = " + start);
      System.out.println("   end = " + end);
      return sum;
} // end of method findSum
```

start and **end** are changed

```
public static void main( String[] args )
{
   int answer;
   int xray = 93;
   int yoke = 104;

   System.out.print("main: xray = " + xray);
   System.out.println("   yoke = " + yoke);

   answer = 17 * findSum( xray, yoke);

   System.out.print("main: xray = " + xray);
   System.out.println("   yoke = " + yoke);

   System.out.println("main: answer is " + answer);
} // end of method main
} // end of class Sums3
```

xray and **yoke** are not changed

Method Example:

```
import java.util.Scanner;
public class Iterations {
   public static void main(String[] args)    {
      Scanner inputScan = new Scanner( System.in );
      int i, number;
      String word;
      do {
         System.out.print("Enter number of iterations (0 to stop): ");
         number = inputScan.nextInt();
      } while ( number < 0 );
      // Print word until user gets tired of it :-)
      while ( number != 0 ) {
         System.out.print("Word to print: ");
         word = inputScan.next();
         for (i = 0; i < number; i++)
            System.out.println( word );
         System.out.println();
         // Get iterations from user. Do not want a negative answer
         do {
            System.out.print("Enter number of iterations (0 to stop): ");
            number = inputScan.nextInt();
         } while ( number < 0 );
      }
   }
} // end of class Iterations
```

Method Example (continued):

```
import java.util.Scanner;
public class IterationsAgain {
   public static int getCount( Scanner inputScan )
   {
      int number;
      do {
         System.out.print("Enter number of iterations (0 to stop): ");
         number = inputScan.nextInt();
      } while ( number < 0 );
      return number;
   } // end of method getCount

   public static void main(String[] args)
   {
      Scanner inputScan = new Scanner( System.in );
      int i, number;
      String word;
      number = getCount( inputScan );
      // Print word until user gets tired of it :-)
      while ( number != 0 ) {
         System.out.print("Word to print: ");
         word = inputScan.next();
         for (i = 0; i < number; i++)
            System.out.println( word );
         System.out.println();

         number = getCount( inputScan );
      }
   }
} // end of class IterationsAgain
```

Method Example:

```
public class ReturnGrade
{
   public static void main( String[] args )
   {
      System.out.println("The grade is " + getGrade(78.5));
      System.out.println("The grade is " + getGrade(93.75));
   } // end of method main

   public static char getGrade( double currentGrade )
   {
      char result;

      if ( currentGrade >= 92.0 )
         result = 'A';
      else if ( currentGrade >= 80.0 )
         result = 'B';
      else if ( currentGrade >= 70.0 )
         result = 'C';
      else if ( currentGrade >= 60.0 )
         result = 'D';
      else
         result = 'E';

      return result;
   } // end of method getGrade
} // end of class ReturnGrade
```

A method that

- has one **`double`** argument, and
- returns a **`char`**.

Method Example (continued):

```
public class ReturnGradeAgain
{
   public static void main( String[] args )
   {
      System.out.println("The grade is " + getGrade(78.5));
      System.out.println("The grade is " + getGrade(93.75));
   } // end of method main

   public static char getGrade( double currentGrade )
   {
      if ( currentGrade >= 92.0 )
         return 'A';
      else if ( currentGrade >= 80.0 )
         return 'B';
      else if ( currentGrade >= 70.0 )
         return 'C';
      else if ( currentGrade >= 60.0 )
         return 'D';
      else
         return 'E';
   } // end of method getGrade

} // end of class ReturnGradeAgain
```

The **return** statement can appear multiple places in the method.

- Here, it is in each branch of the **if** statement.
- When a **return** is executed, the method is finished.