Introduction to Methods
Part 1: Defining a Method

Course: CPSC 1150

Instructor: Dr. Bita Shadgar

Lecture 11

Learning Outcomes

- Realize the benefit of using methods in programming
- Apply and call a method
- Explain how the control is moving between caller and calling method
- Recognize the different part of a method
- Define and declare a method

Problem: finding the sum of a range

Finding the sum of a range

Write a program that calculates the sum of integers from 1 to 10, from 20 to 37 and from 35 to 49.

```
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 <= 37; i++)
  sum += i;
System.out.println("Sum from 20 to 37 is " + sum);

sum = 0;
for (int i = 35; i <= 49; i++)
  sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

Performing repetitive tasks

- Computers are good at repetitive tasks, but humans are not
- How can we avoid writing pages and pages of code?
  - One way is with loops
- Some types of repetition **don't work** well with loops. For example, using the same code:
  - In different programs
  - With unrelated inputs (e.g. finding the sum of a range)

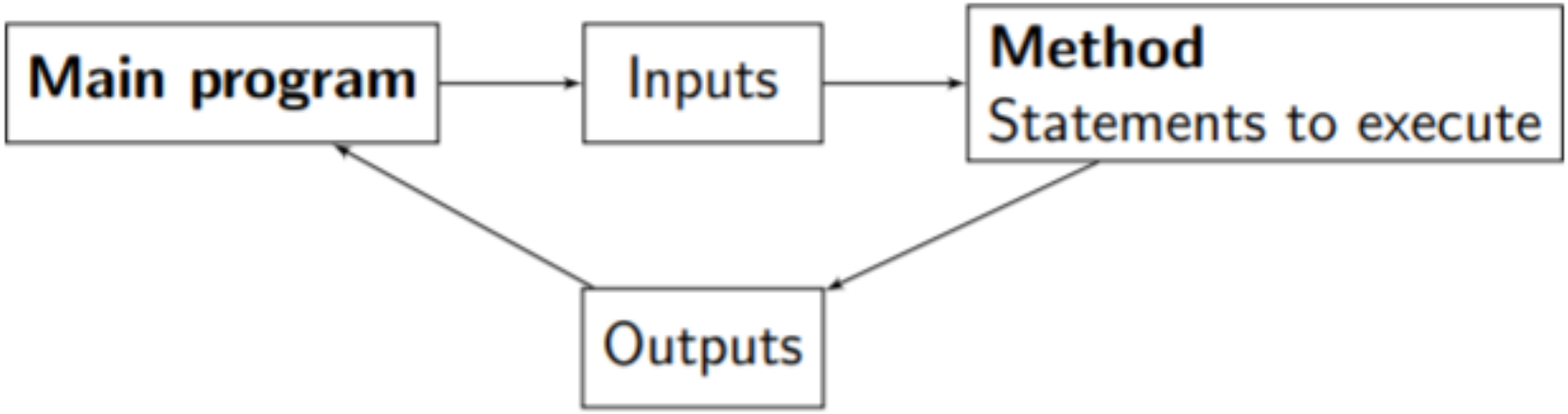
Examples of methods you already know

You have been using methods all along! For example:

- `System.out.println("Hello world!");`
  - Input is a String to print.
  - Action performed is to print the String to the console.
  - Output is void – this means nothing is returned.
- `num1 = input.nextInt();`
  - Input is void (empty).
  - Action performed is to read the next int typed by the user.
  - Output is the int that the user typed – this output is stored in the variable num1.
- These are both examples of **calls** to methods.
  - Neither example contains the actual method **declaration** – the code id used when a method is invoked (called).

What is a method?

- A named group of statements that takes an input from a program, performs some actions (process), and returns an output.
- Called “functions” or “procedures” in other programming languages.



What happens when a method is used?

When a program **invokes** (or calls) a method:

- The program passes arguments (data in the form of constants or variables) to the method, and passes control to the method.
- The method executes a block of statements, using the arguments that were passed in. Methods can call other methods, and can even call themselves! We will see this later.
- The method returns a value to the program (or not, if it is a void method) and then terminates, returning control to the program.
- The program resumes where it left off, and can now use the value returned by the method.

Why use methods?

- Reusable across different pieces of software
- Help us see a problem as a collection of subproblems
- Not too many local variables (method code is isolated from main program)
- Don’t need to know how a method works to use it: Just need to know input/output type(s) and range, and what the function is used for
- Easier to test each method in isolation than the whole program at once
- Convenience of method overloading (we will see this later)
- Can give programmer control over how variables are changed by a user, through encapsulation (not in this course)
- And more...

Sum of a range: Subproblems

Recall:

Finding the sum of a range

Write a program that calculates the sum of integers from 1 to 10, from 20 to 37 and from 35 to 49.

Subproblems:

- Communicate instructions to the user.
- Accept user input in a specific range.
- Calculate the sum of integers in a range
- Print the sum to the console for different ranges.

We will focus on the third item.

Sum of a range : How could we use a function?

If we want to...

– Calculate the sum of integers in a range

We could have a function that...

Given the start and end of the range, calculates and returns the sum

Our function: findSum()

- Input: start and end of the range
- Process: adds up the numbers in this range
- Output: the sum

main function – make 3 intervals

Assume user inputs are valid.

– Create a **for loop** to generate different range

main function

```
public static void main(String []args) {
   Scanner input = new Scanner(System.in);
   final int INTERVALS = 3;
   int start, end, sum;
   for(int i=0; i<INTERVALS ; i++){









   } //end for
} //end main
```

main function – getting user inputs

- In the loop we need to get the start and end of each intervals from user

main function

```
public static void main(String []args) {
   Scanner input = new Scanner(System.in);
   final int INTERVALS = 3;
   int start, end, sum;
   for(int i=0; i<INTERVALS ; i++){
      System.out.print("Enter start and end of range: ");
      start = input.nextInt();
      end = input.nextInt();



   } //end for
} //end main
```

main function – calling function

- Call function to calculate the sum of range.
  - start and end of range are passed to function as arguments
  - The result is returned and saved in sum

main function

```
public static void main(String []args) {
   Scanner input = new Scanner(System.in);
   final int INTERVALS = 3;
   int start, end, sum;
   for(int i=0; i<INTERVALS ; i++){
      System.out.print("Enter start and end of range: ");
      start = input.nextInt();
      end = input.nextInt();
      sum = findSum(start, end);

   } //end for
} //end main
```

main function – display the results

- Print out sum in an appropriate message.

main function

```
public static void main(String []args) {
   Scanner input = new Scanner(System.in);
   final int INTERVALS = 3;
   int start, end, sum;
   for(int i=0; i<INTERVALS ; i++){
      System.out.print("Enter start and end of range: ");
      start = input.nextInt();
      end = input.nextInt();
      sum = findSum(start, end);
      System.out.printf("Sum from %d to %d is %d.\n\n", start, end, sum);
   } //end for
} //end main
```

function definition – function header

Let's look at the different parts of a **function definition**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

- **function header**: used to identify the function
- **Different** from calling or invoking a function, which doesn't include the modifiers or types
  - e.g. of calling a function:

```
theSum = findSum(12, 25);
```

Method declaration - Modifiers

Let's look at the different parts of a **method declaration**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

- **Modifiers**: keywords added to variable or method declarations to change their meanings
- e.g. `public, private, static, abstract`

function definition - return type

Let's look at the different parts of a **function definition**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

- **Return type**: data type of the value that will be returned
- e.g. `int`, `double`, `string`, `void`

function definition - function name

Let's look at the different parts of a **function definition**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

- **function name:** usually a verb, start with lower case, make it descriptive but concise
  - e.g. `getMax`, `calcVolume`, `setColour`
  - Don't do something like: `findLargestNumberInListThenSquareItAndDivideByTwo`

function definition - Parameter list in parentheses

Let's look at the different parts of a **function definition**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

- **Parameter list in parentheses:** each formal parameter is preceded by its data type, and separated by commas (can be empty)
- e.g. `(int ageInDays, float weightInPounds), ()`

Method declaration - Method body

Let's look at the different parts of a **method declaration**.

```
public static int findSum(int s, int e){
   //statements go here
}
```

– **Method body:** enclosed by curly braces, statements that the method will execute

Implementation of findSum

Code for findSum

```
public static int findSum(int s, int e){
    int result = 0;



}
```

– Declare an integer called result and initialized to zero

Implementation of findSum

Code for findSum

```
public static int findSum(int s, int e){
    int result = 0;
    for (int i=s; i<=e; ++i){

    } //end for

}
```

– Use a for loop which executes e-s+1 times

Implementation of findSum

Code for findSum

```
public static int findSum(int s, int e){
    int result = 0;
    for (int i=s; i<=e; ++i){
            result += i;
    } //end for

}
```

– Each iteration of the loop, adds a number in the range to the result

Implementation of findSum

Code for findSum

```
public static int findSum(int s, int e){
    int result = 0;
    for (int i=s; i<=e; ++i){
            result += i;
    }//end for
    return result;
}
```

– Return `result` which is now the sum of all numbers between s and e (inclusive)

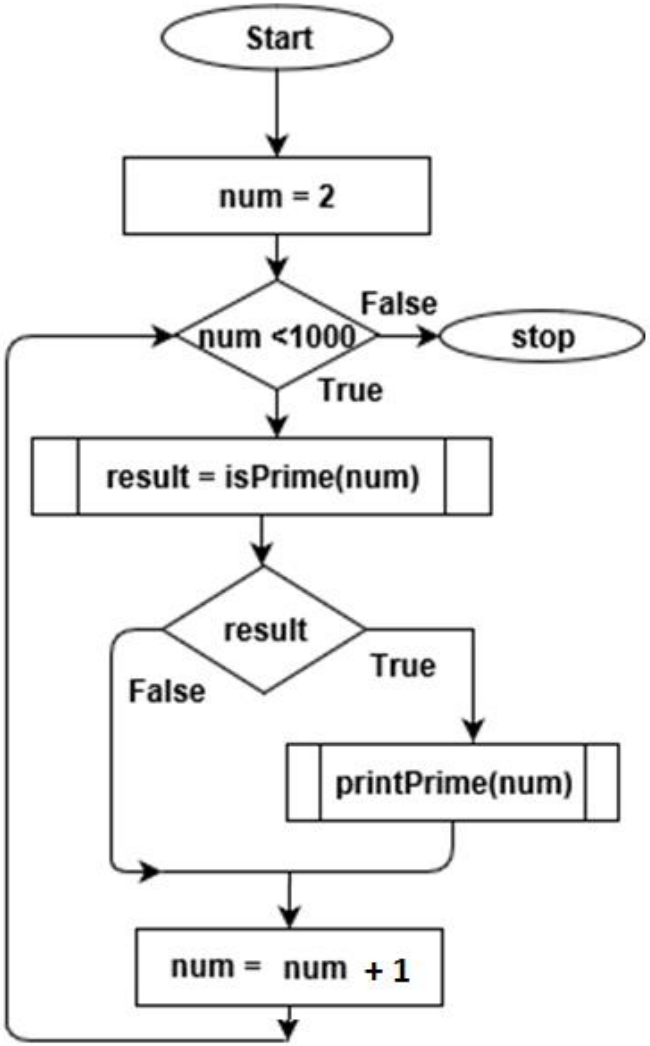
Complete code for FindSum

```
import java.util.Scanner;
public class Test{
   public static void main(String[] args){
      Scanner input = new Scanner(System.in);
      final int INTERVALS = 3;
      int start, end, sum;
      for(int i=0; i<INTERVALS ; i++){
         System.out.print("Enter start and end of range: ");
         start = input.nextInt();
         end = input.nextInt();
         sum = findSum(start, end); //calling function
         System.out.printf("Sum from %d to %d is %d.\n\n", start, end, sum);
      } //end for
   }
   // defining function
   public static int findSum(int s, int e){
      int result = 0;
      for (int i=s; i<=e; ++i){
          result += i;
      } //end for
      return result;  //return value
   }
}
```

More Practice

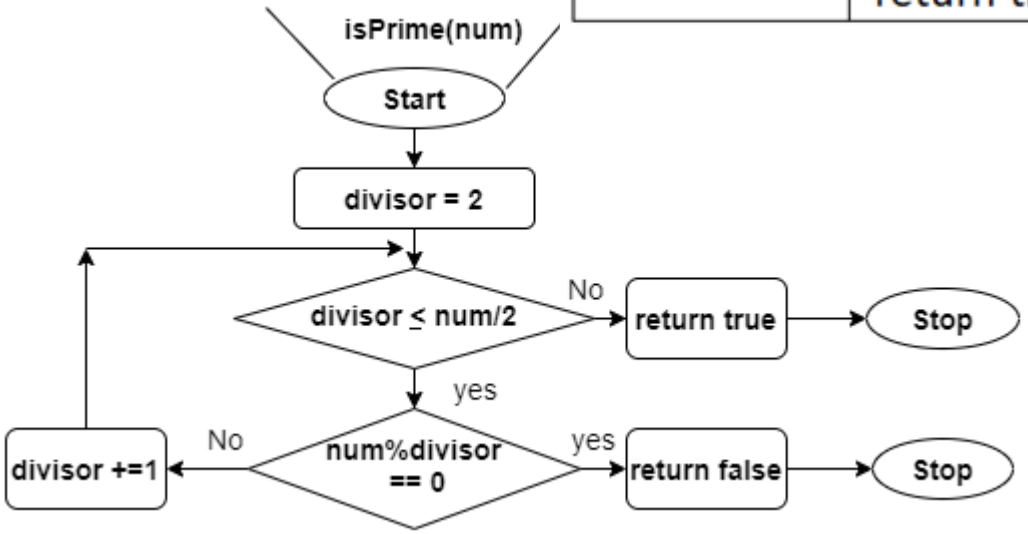
Display prime numbers

Use functions to write a program that finds and displays the prime numbers between 2 and 1000.



isPrime function

IPO chart for `isPrime` function		
Input	**Processing**	**Output**
An integer number, named num	Check if the input is prime or not	true if number is prime, otherwise false
	Divide num by numbers from 2 up to num/2, if the remainder is zero, return false Otherwise, num is Prime and return true.	



printPrime function

IPO chart for `printPrime` function		
Input	**Processing**	**Output**
An integer number, named num	Prints the input	nothing