User-defined methods and classes

Method basics

- A **method definition** is a named list of statements (block of code), that also includes the type (see below)
- A method's statements are executed when a method is **called**.
- When a method ends (or returns), the flow of control returns to where the method was last called.

- A method definition must be part of a class
- A method is *public* and *static* if it can be called directly from the main program (Note: main itself is a method).
- A method is *void* if it does not return a value

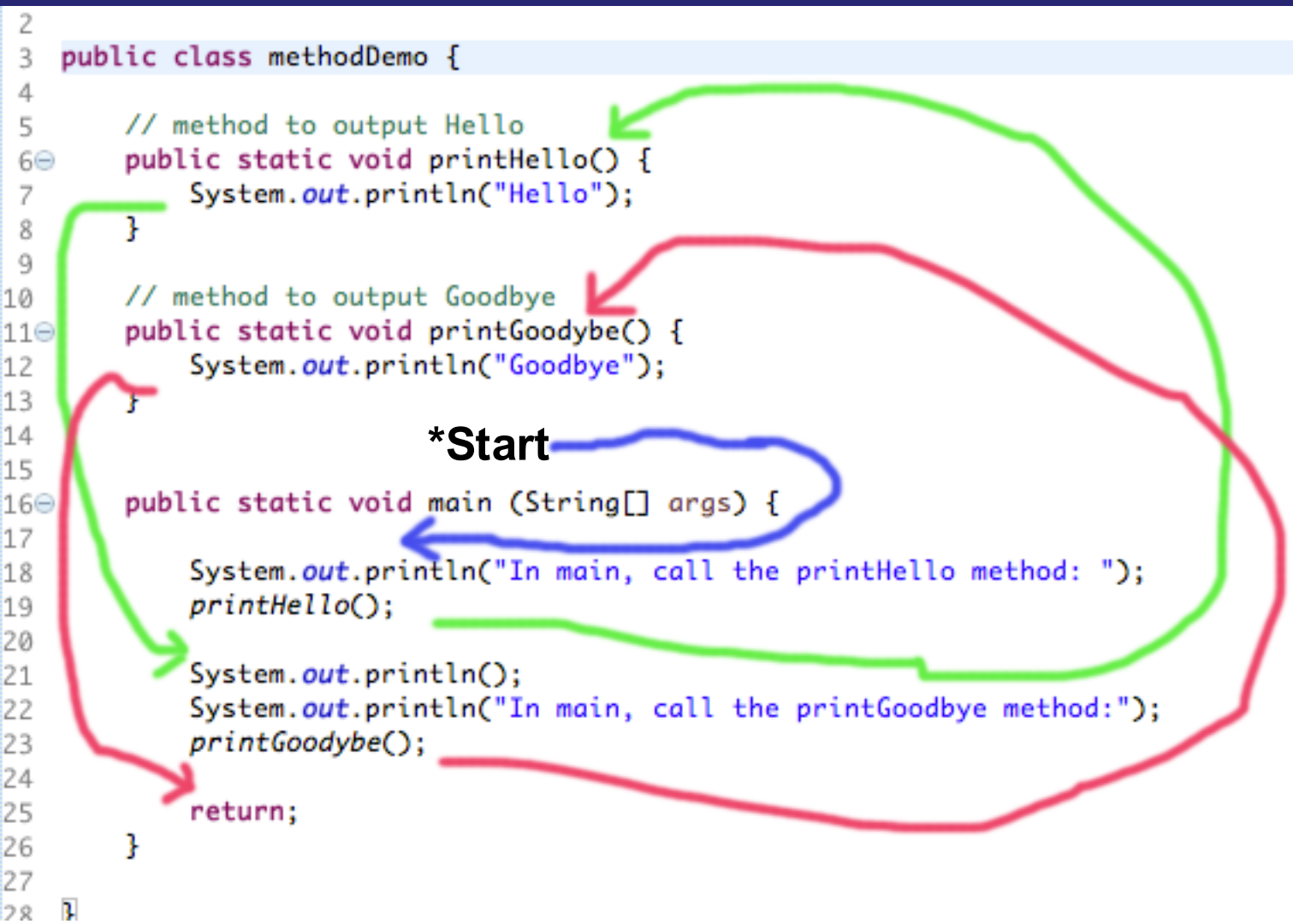
Characterization of methods

- Methods that do not return a value are *void*
- *Integer* methods return integer values, *double* methods return double values, etc.
- Any method can take inputs. Method *parameters* are variables defined in the method header that correspond to these inputs. When the method is called, these inputs are known as *arguments*.

Method example

```
2
3  public class methodDemo {
4
5      // method to output Hello
6      public static void printHello() {
7          System.out.println("Hello");
8      }
9
10     // method to output Goodbye
11     public static void printGoodybe() {
12         System.out.println("Goodbye");
13     }
14
15
16     public static void main (String[] args) {
17
18         System.out.println("In main, call the printHello method: ");
19         printHello();
20
21         System.out.println();
22         System.out.println("In main, call the printGoodbye method:");
23         printGoodybe();
24
25         return;
26     }
27
28 }
```

Method example




```
2
3  public class methodDemo {
4
5      // method to output Hello
6      public static void printHello() {
7          System.out.println("Hello");
8      }
9
10     // method to output Goodbye
11     public static void printGoodybe() {
12         System.out.println("Goodbye");
13     }
14
15
16     public static void main (String[] args) {
17
18         System.out.println("In main, call the printHello method: ");
19         printHello();
20
21         System.out.println();
22         System.out.println("In main, call the printGoodbye method:");
23         printGoodybe();
24
25         return;
26     }
27
28 }
```

Method with parameters

```
public class returnDemo{

    // a method that adds two numbers and returns the sum
    public static int add(int x, int y) {
        int total = x + y;
        return total;
    }

    public static void main (String[] args) {

        Scanner scnr = new Scanner(System.in);
        System.out.print("Enter any two integers: ");
        int num1 = scnr.nextInt();
        int num2 = scnr.nextInt();

        int sum = add(num1, num2);
        System.out.println("The sum of " + num1 + " and " + num2 + "  is: " + sum);

        // Alternatively, we could call the method in the System.out.println statement
        // System.out.println("The sum of " + num1 + " and " + num2 + "  is: " + add(num1, nu

        scnr.close();
        return;
    }
}
```

x and *y* are method **parameters**

num1 and *num2* are ***arguments*** whose values are passed to the corresponding method parameters

Method with parameters

```
public class returnDemo{

    // a method that adds two numbers and returns the sum
    public static int add(int x, int y) {
        int total = x + y;
        return total;
    }

    public static void main (String[] args) {

        Scanner scnr = new Scanner(System.in);
        System.out.print("Enter any two integers: ");
        int num1 = scnr.nextInt();
        int num2 = scnr.nextInt();

        int sum = add(num1, num2);
        System.out.println("The sum of " + num1 + " and " + num2 + "  is: " + sum);

        // Alternatively, we could call the method in the System.out.println statement
        // System.out.println("The sum of " + num1 + " and " + num2 + "  is: " + add(num1, nu

        scnr.close();
        return;
    }
}
```

value of num1 passed to x

value of num2 passed to y

*Start

Reasons for methods

1. Improves program readability
2. Allows for modular (piece-wise) program development
3. Allows for collaborative program development
4. Reduces the writing of redundant code

Additional Tips regarding Methods

- Method **stubs** are useful for writing a program that has methods (see *methodStub.java* example)
- Don't forget about **scope**. A variable declared within a method does not exist outside of it. Similarly, a variable declared in *main* (which is a method) will not exist in other methods.
- In general, methods do not change the value of any arguments passed to them (but arrays elements are an exception; see MethodsWithArrays*.java* example)
- If a method needs to use a scanner, declare a static scanner at the top of your class

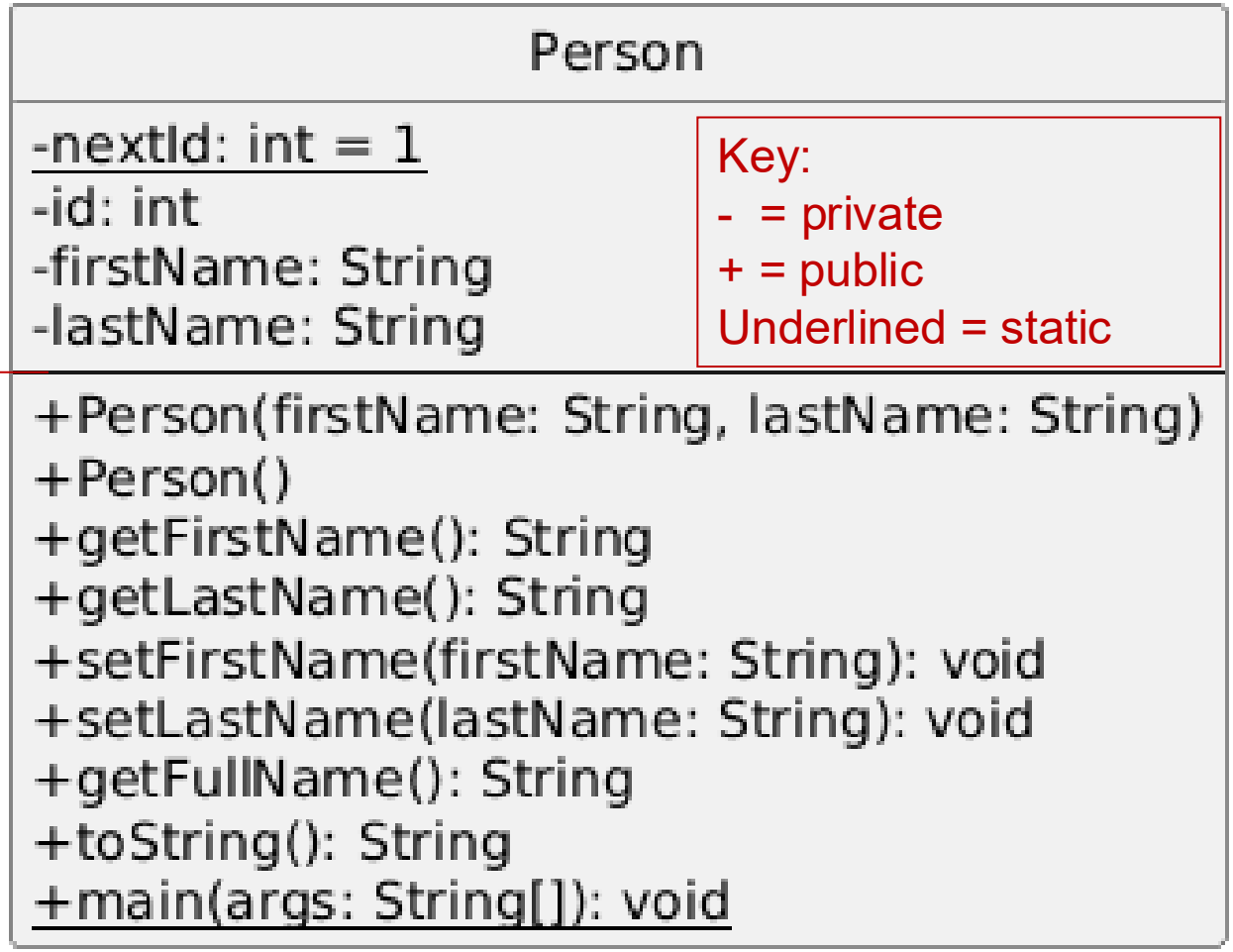
Classes

- A class defines an object that can contain
  - fields: variables/objects that hold data
  - methods: actions or behavior, generally acting on its data
  - constructors: determine initialization of object
- Members of classes can be:
  - Public: accessible from any class
  - Private: accessible only from within the same class
  - Static: belong to class, and not an instance of the object

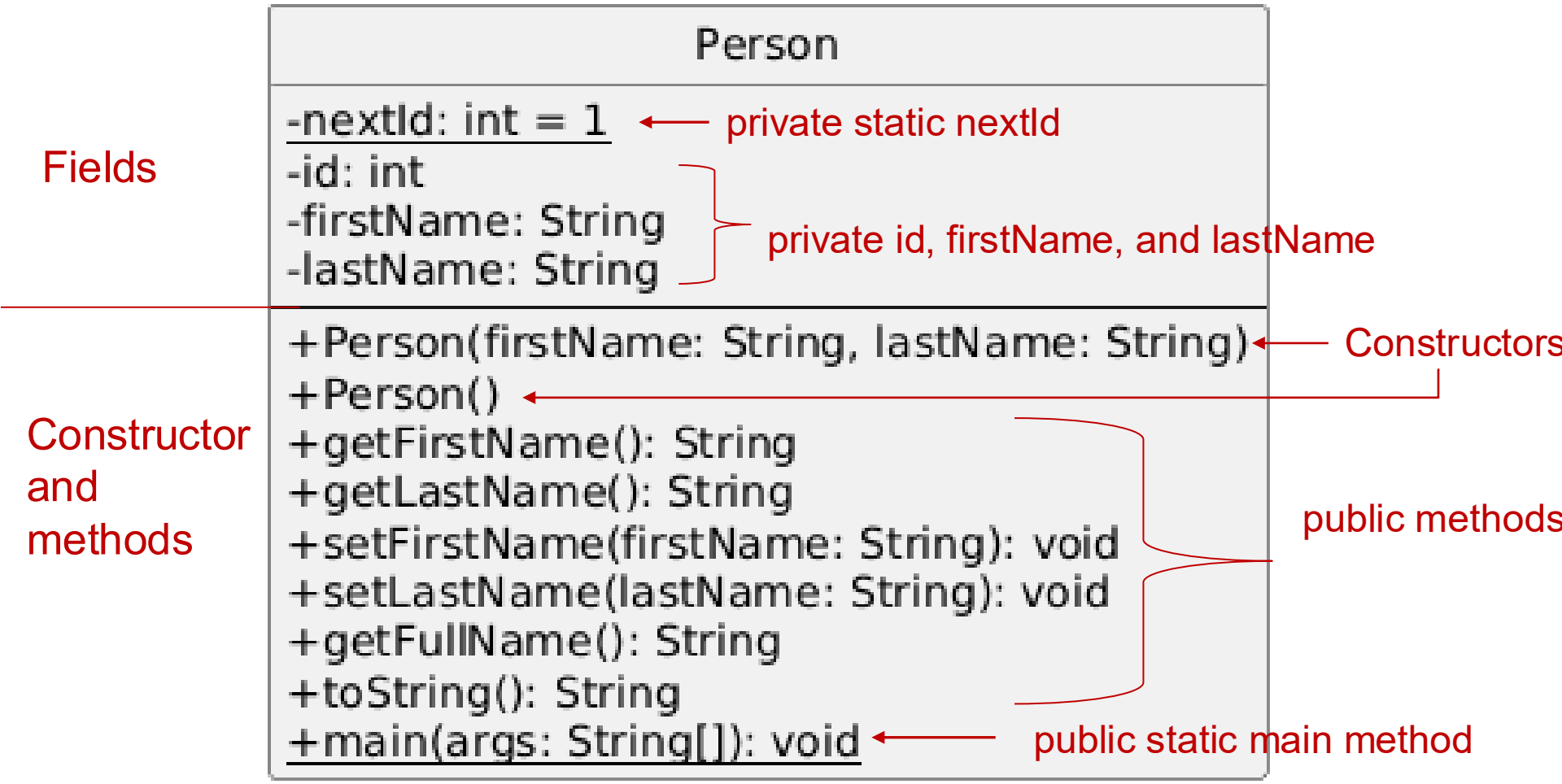
Unified Modeling Language (UML) diagram

Fields

Constructor and methods



Unified Modeling Language (UML) diagram



Good Object-Oriented Programming Principles

- A class should represent only 1 concept
- Use *encapsulation* to hide internal data and state
- Make sure the state is always valid
- Use *abstraction* to expose the *what* and hide the *how*
- Make objects immutable (final) if possible
- Naming conventions:
  - Nouns for class names
  - Verbs for methods
- More to come…