
Structured Programming

Lecture 3

Dr. Obead Alhadreti

Outline

- ▶ Constructors
- ▶ **this** keyword

Constructors

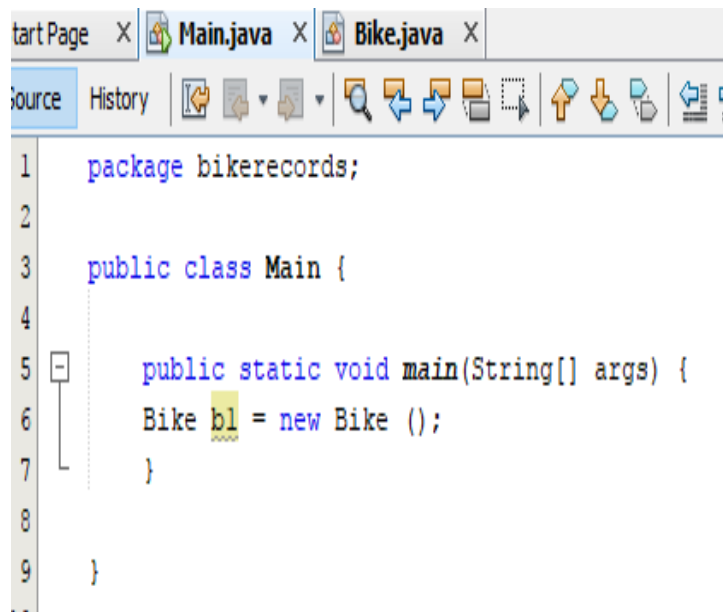
Constructor

- ▶ **Constructor:** is a *special type of method* that is used to initialize the object.
- ▶ It is like a method in that it can have an access modifier (like public or private), a name, parameters, and executable code.
- ▶ **Constructors have the following special features:**
 1. Constructors have the same name as the class in which they are defined, and typically should be **public**.
 2. Constructors cannot have a **return type**: not even **void**.

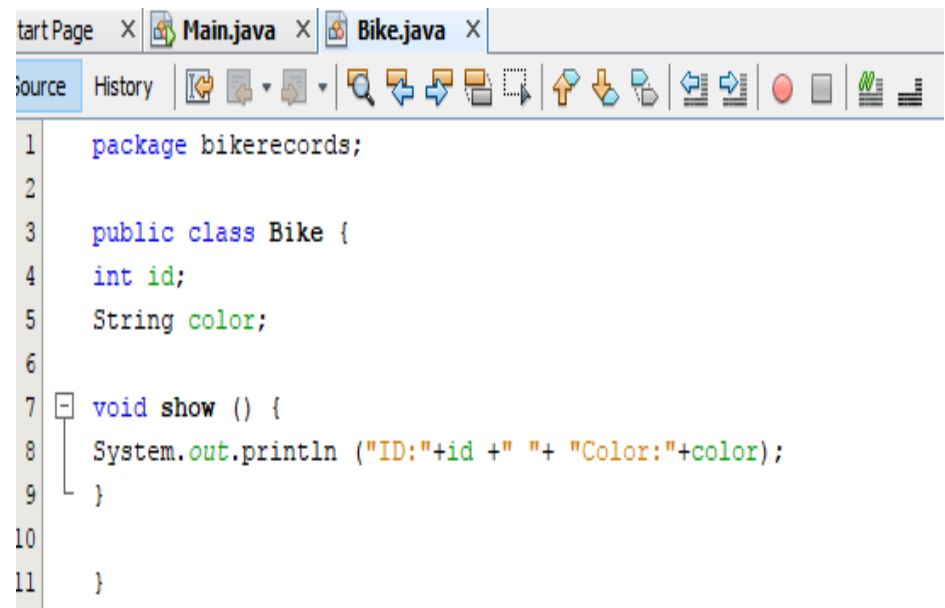
Constructor

- ▶ You must call a constructor to create each object.

Bike **b1** = new **Bike ()** ; ← constructor



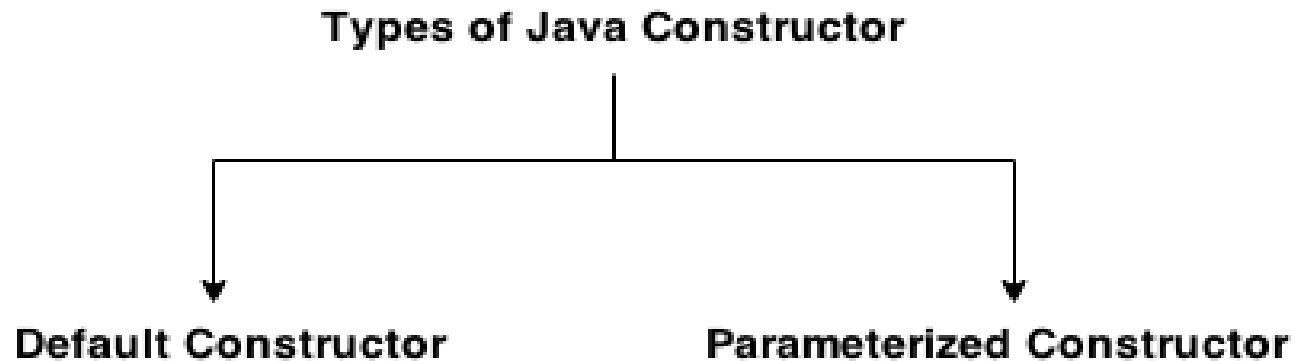
```
1 package bikerecords;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Bike b1 = new Bike ();
7     }
8
9 }
```



```
1 package bikerecords;
2
3 public class Bike {
4     int id;
5     String color;
6
7     void show () {
8         System.out.println ("ID:"+id +" "+ "Color:"+color);
9     }
10
11 }
```

Constructor

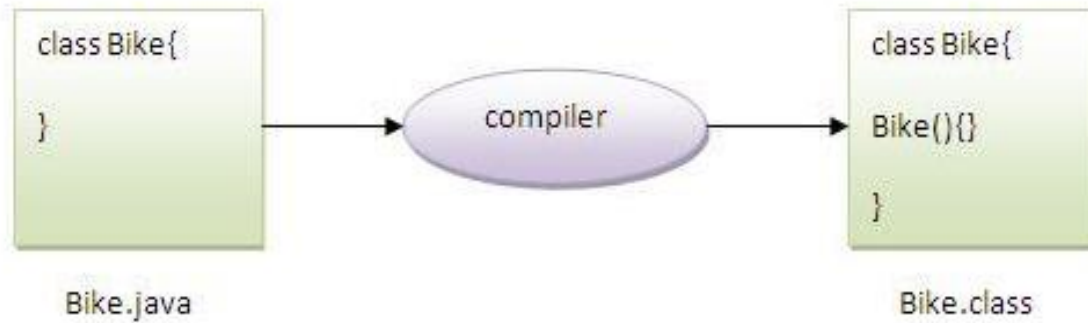
- There are two types of constructors:



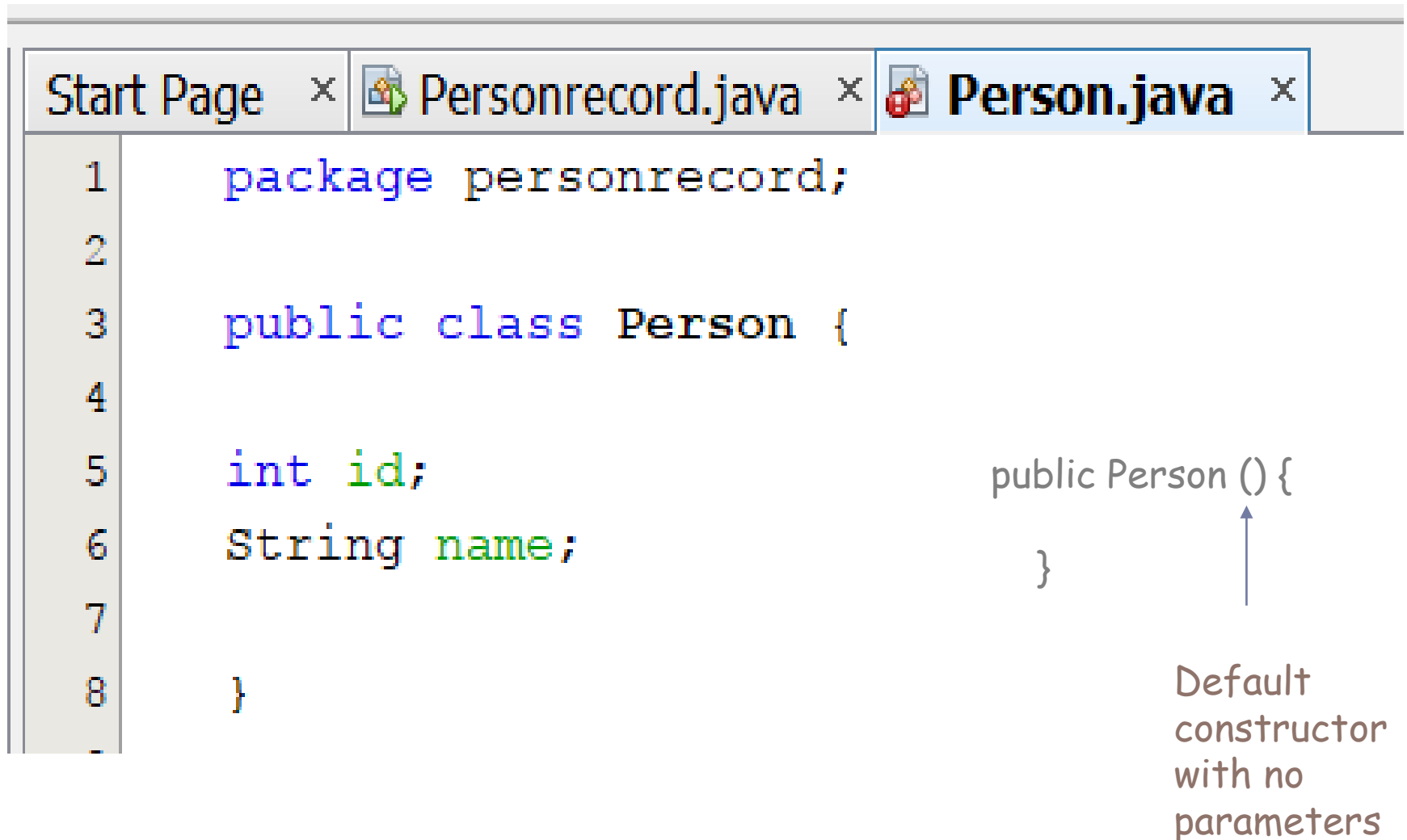
1. Default Constructor

- ▶ If a class does not define constructors, the java compiler **automatically** provides a default constructor with no parameters (local variables).
- ▶ A constructor with no parameter is called the *default constructor*.
- ▶ The default constructor provides the default values for an object.

1. Default Constructor



Example of default constructor



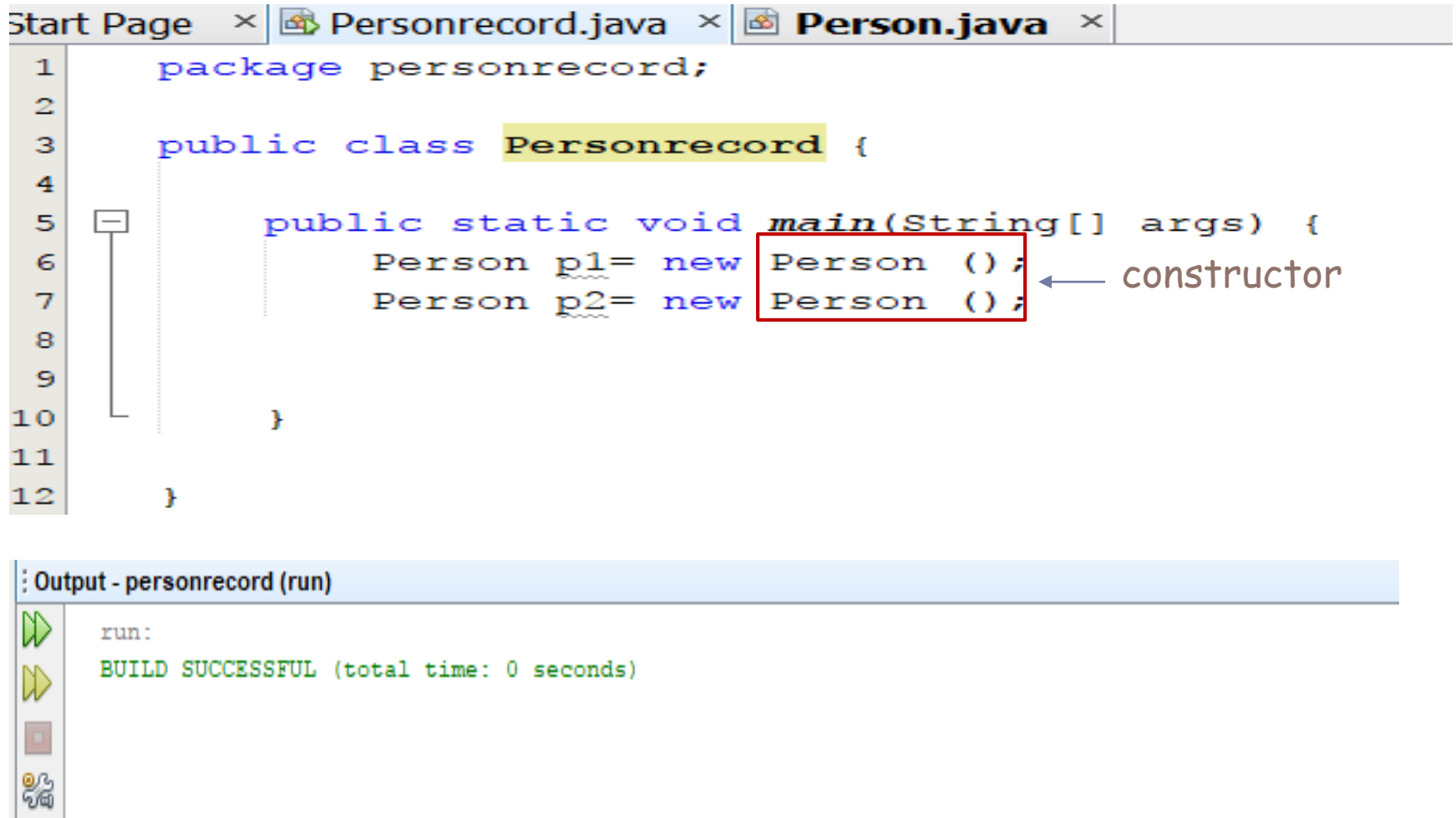
The image shows a screenshot of an IDE window titled "Person.java". The code defines a package "personrecord" and a public class "Person". Inside the class, there are two attributes: "int id;" and "String name;". A default constructor "public Person () {" is shown to the right of the class definition, with an arrow pointing to it from the text "Default constructor with no parameters".

```
1 package personrecord;
2
3 public class Person {
4
5     int id;
6     String name;
7
8 }
```

```
public Person () {
}
```

Default constructor with no parameters

Example of default constructor



The screenshot shows an IDE with two tabs: 'Personrecord.java' and 'Person.java'. The 'Personrecord.java' tab is active, displaying the following code:

```
1 package personrecord;
2
3 public class Personrecord {
4
5     public static void main(String[] args) {
6         Person p1= new Person ();
7         Person p2= new Person ();
8     }
9 }
10
11
12
```

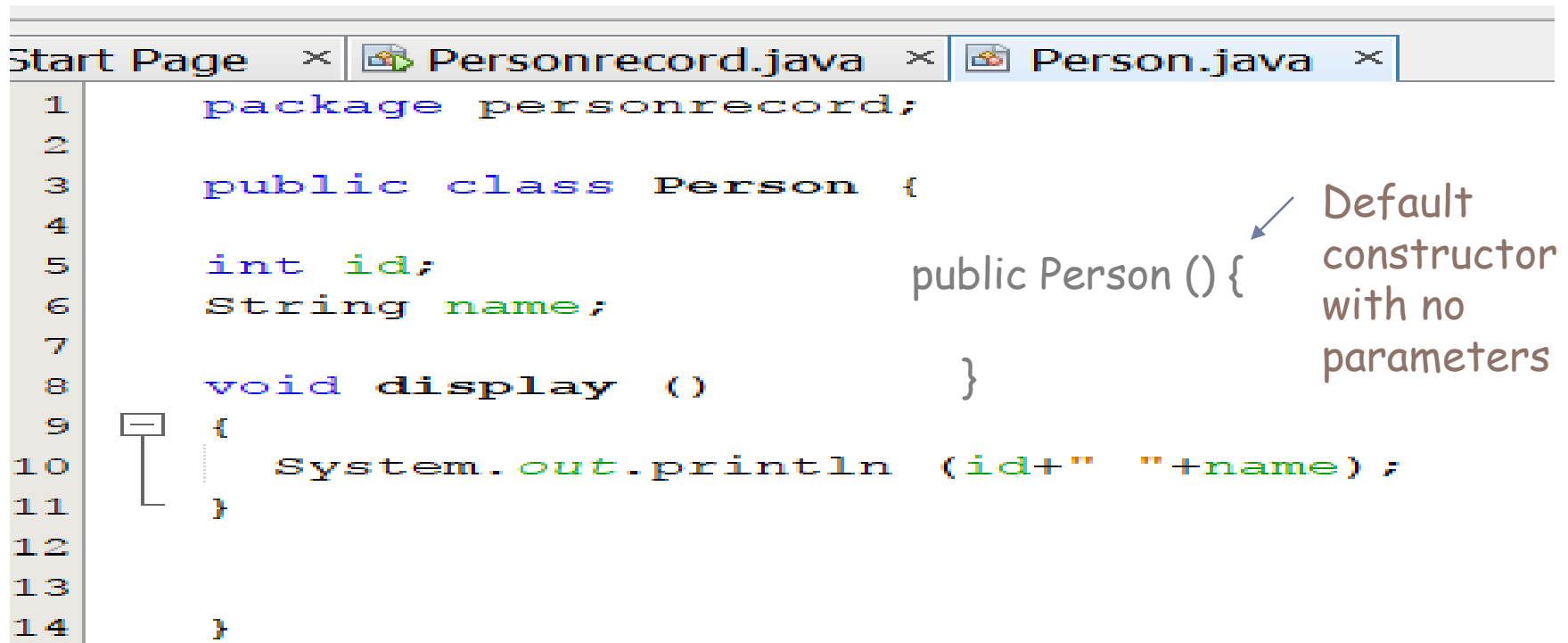
The code is annotated with a red box around the `new Person ()` expressions on lines 6 and 7, and an arrow pointing to them with the text 'constructor'. The 'Person.java' tab is also visible but its content is not shown.

Below the code editor, the 'Output - personrecord (run)' window shows the execution results:

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example of default constructor that displays the default values

- ▶ Default constructors can also be used to show the default values to the object like 0, null, etc depending on the type

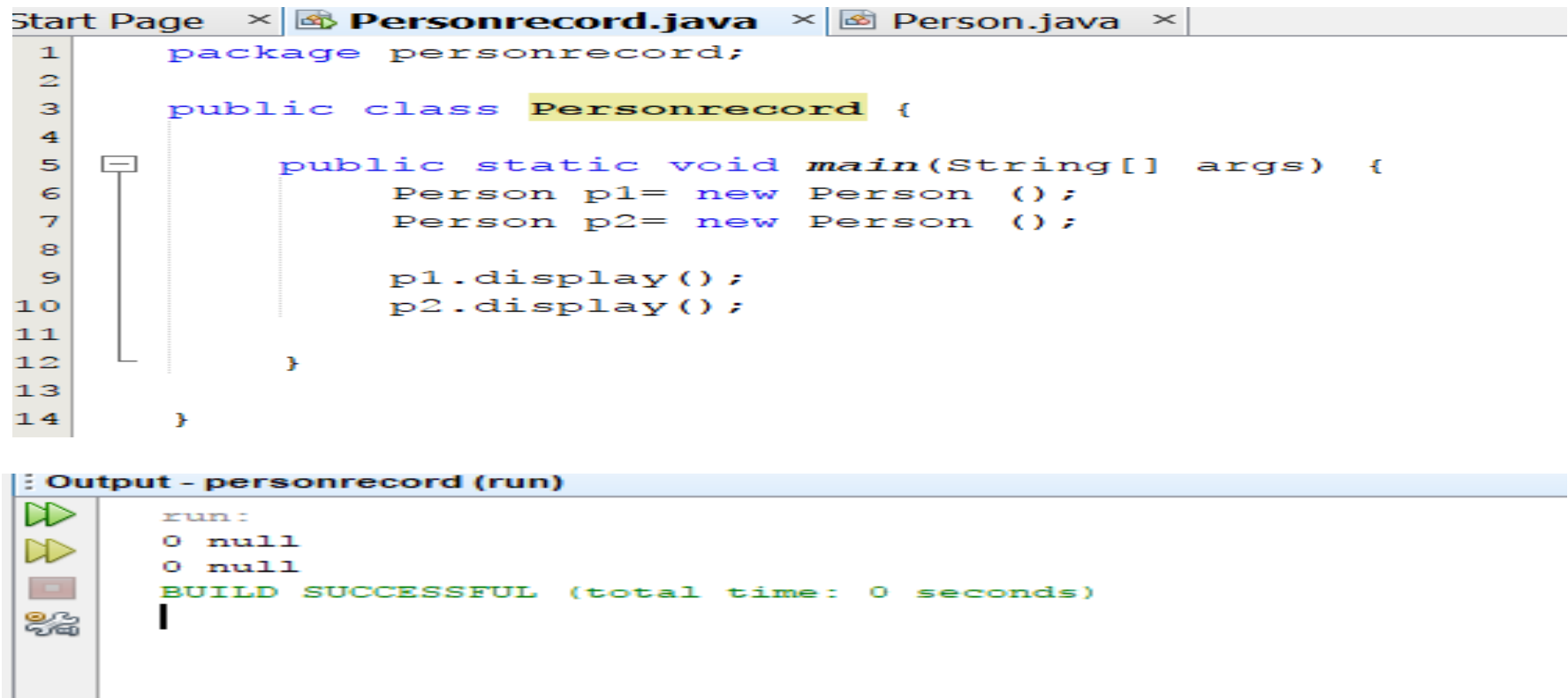


The screenshot shows a Java IDE with three tabs: 'Start Page', 'Personrecord.java', and 'Person.java'. The 'Person.java' tab is active, displaying the following code:

```
1 package personrecord;
2
3 public class Person {
4     int id;
5     String name;
6
7     void display ()
8     {
9         System.out.println (id+" "+name);
10    }
11
12
13
14 }
```

An annotation 'public Person () {' is placed next to the code, with an arrow pointing to it from the text 'Default constructor with no parameters'.

Example of default constructor that displays the default values



The screenshot shows an IDE with two tabs: 'Personrecord.java' and 'Person.java'. The 'Personrecord.java' tab is active, displaying the following code:

```
1 package personrecord;
2
3 public class Personrecord {
4
5     public static void main(String[] args) {
6         Person p1= new Person ();
7         Person p2= new Person ();
8
9         p1.display();
10        p2.display();
11
12    }
13
14 }
```

Below the code editor, the 'Output - personrecord (run)' window is visible, showing the execution results:

```
run:
0 null
0 null
BUILD SUCCESSFUL (total time: 0 seconds)
```

- ▶ In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

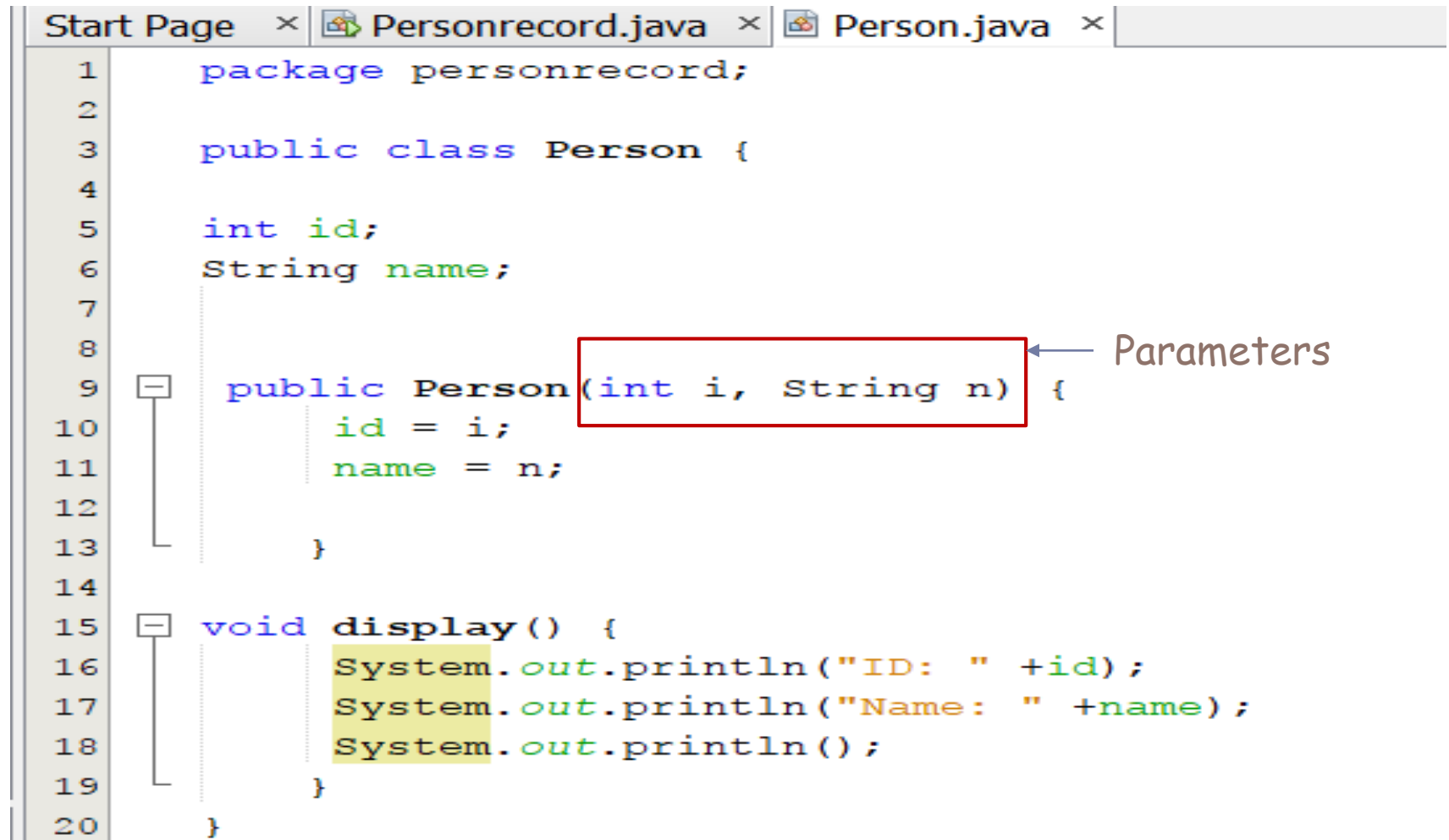
2. Java parameterized constructor

- ▶ A constructor that have parameters is known as parameterized constructor.
- ▶ Why to use parameterized constructor?
Parameterized constructor is used to provide different values to distinct objects.
- ▶ In case you create a constructor, then java will **not** create a default constructor.

Example 1 of parameterized constructor

- ▶ In this example, we have created a parametrized constructor of Person class that have **two parameters**.
- ▶ We can have any number of parameters in the constructor.

Example 1 of parameterized constructor

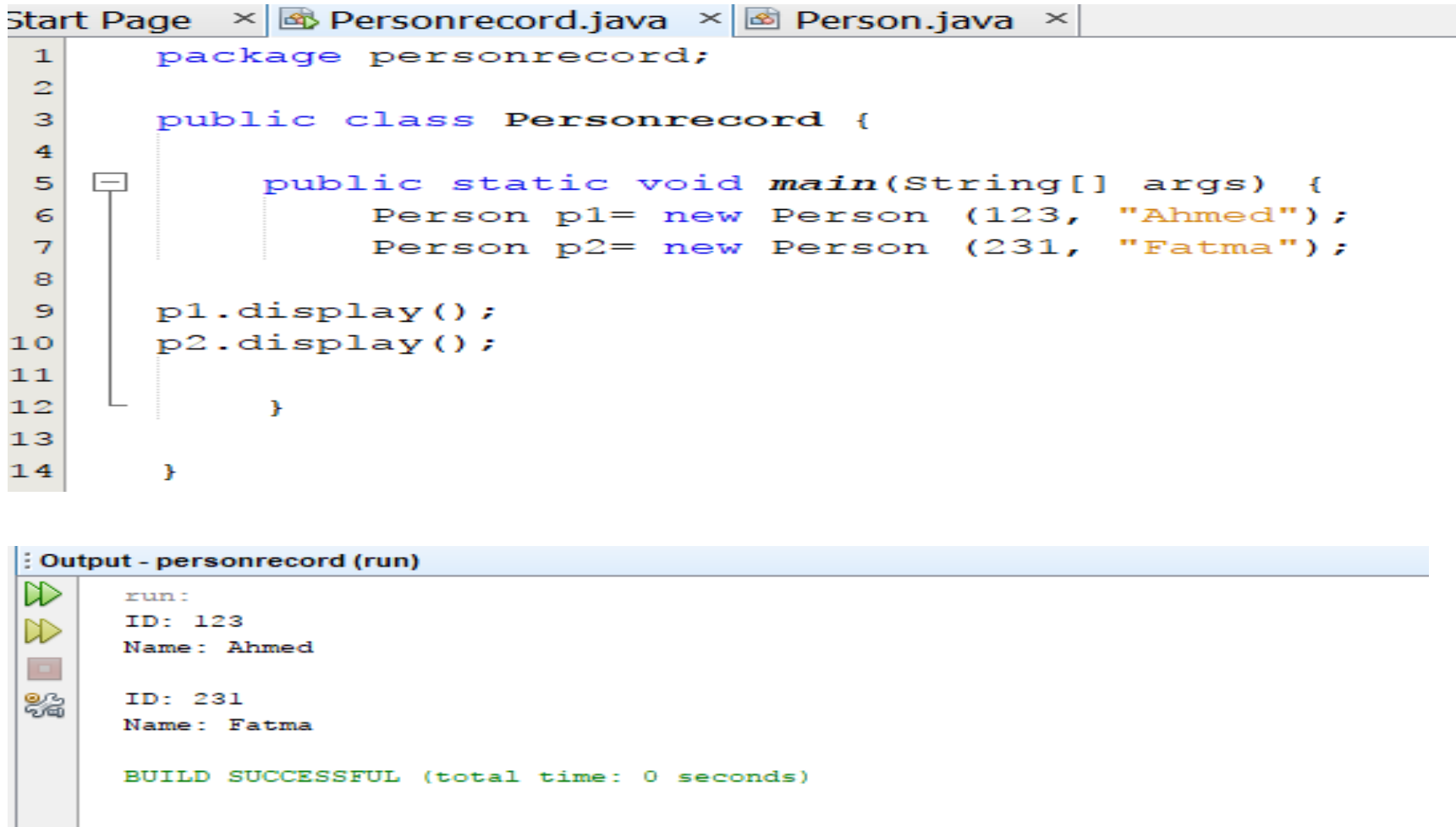


```
Start Page × Personrecord.java × Person.java ×
1 package personrecord;
2
3 public class Person {
4
5     int id;
6     String name;
7
8
9     public Person(int i, String n) {
10         id = i;
11         name = n;
12     }
13
14
15     void display() {
16         System.out.println("ID: " + id);
17         System.out.println("Name: " + name);
18         System.out.println();
19     }
20 }
```

Parameters

The image shows a Java IDE with three tabs: 'Start Page', 'Personrecord.java', and 'Person.java'. The 'Personrecord.java' tab is active, displaying the following code: Line 1: 'package personrecord;'. Line 2: an empty line. Line 3: 'public class Person {'. Line 4: an empty line. Line 5: 'int id;'. Line 6: 'String name;'. Line 7: an empty line. Line 8: an empty line. Line 9: 'public Person(int i, String n) {'. Line 10: 'id = i;'. Line 11: 'name = n;'. Line 12: '}'. Line 13: an empty line. Line 14: an empty line. Line 15: 'void display() {'. Line 16: 'System.out.println("ID: " + id);'. Line 17: 'System.out.println("Name: " + name);'. Line 18: 'System.out.println();'. Line 19: '}'. Line 20: '}'. A red rectangular box highlights the constructor signature 'public Person(int i, String n) {'. A blue arrow points from the text 'Parameters' to this box. On the left side of the code editor, there are two fold icons (minus signs in squares) corresponding to the constructor and the display method. The constructor's body (lines 10-12) is visible, and the display method's body (lines 16-18) is highlighted in yellow.

Example 1 of parameterized constructor



The screenshot displays an IDE with two tabs: 'Personrecord.java' and 'Person.java'. The 'Personrecord.java' tab is active, showing the following code:

```
1 package personrecord;
2
3 public class Personrecord {
4
5     public static void main(String[] args) {
6         Person p1= new Person (123, "Ahmed");
7         Person p2= new Person (231, "Fatma");
8
9         p1.display();
10        p2.display();
11
12    }
13
14 }
```

Below the code editor, the 'Output - personrecord (run)' window shows the execution results:

```
run:
ID: 123
Name: Ahmed

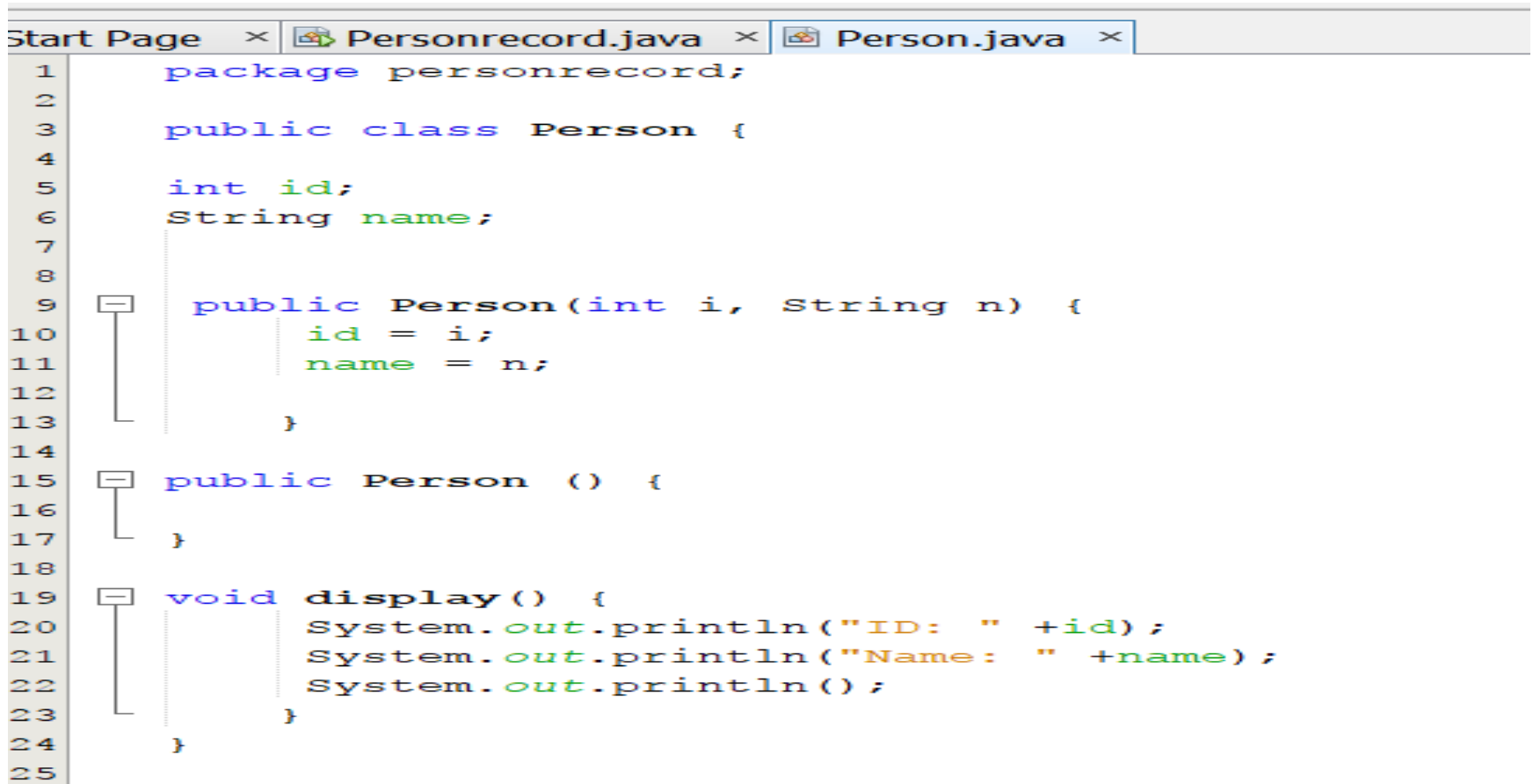
ID: 231
Name: Fatma

BUILD SUCCESSFUL (total time: 0 seconds)
```

Example 2 of parameterized constructor

- ▶ In this example, we have created the parametrized constructor of Person class that have two parameters.
- ▶ We have also created a **constructor** with no parameters to create **object p3** which will have no initial values. It is like we have used a **default constructor**. We have created this constructor because the default constructor is no longer used by java.

Example 2 of parameterized constructor



The screenshot shows a Java IDE with two tabs: 'Personrecord.java' and 'Person.java'. The 'Person.java' tab is active, displaying the following code:

```
1 package personrecord;
2
3 public class Person {
4
5     int id;
6     String name;
7
8
9     public Person(int i, String n) {
10         id = i;
11         name = n;
12     }
13
14
15     public Person () {
16
17     }
18
19     void display() {
20         System.out.println("ID: " +id);
21         System.out.println("Name: " +name);
22         System.out.println();
23     }
24 }
25
```

Example 2 of parameterized constructor

```
Start Page x Personrecord.java x Person.java x
1 package personrecord;
2
3 public class Personrecord {
4
5     public static void main(String[] args) {
6         Person p1 = new Person (123, "Ahmed");
7         Person p2 = new Person (231, "Fatma");
8         Person p3 = new Person ();
9
10        p3.id = 211;
11        p3.name = "Nourah";
12
13        p1.display();
14        p2.display();
15        p3.display ();
16
17    }
18
19 }
```

```
Output - personrecord (run)
run:
ID: 123
Name: Ahmed

ID: 231
Name: Fatma

ID: 211
Name: Nourah

BUILD SUCCESSFUL (total time: 0 seconds)
```

Example 2 of parameterized constructor

```
art Page x | Personrecord.java x | Person.java x |
package personrecord;

public class Personrecord {

    public static void main(String[] args) {
        Person p1 = new Person (123, "Ahmed");
        Person p2 = new Person (231, "Fatma");
        Person p3 = new Person ();

        p1.display();
        p2.display();
        p3.display ();
    }
}
```

Output - personrecord (run)

```
run:
ID: 123
Name: Ahmed

ID: 231
Name: Fatma

ID: 0
Name: null

BUILD SUCCESSFUL (total time: 0 seconds)
```

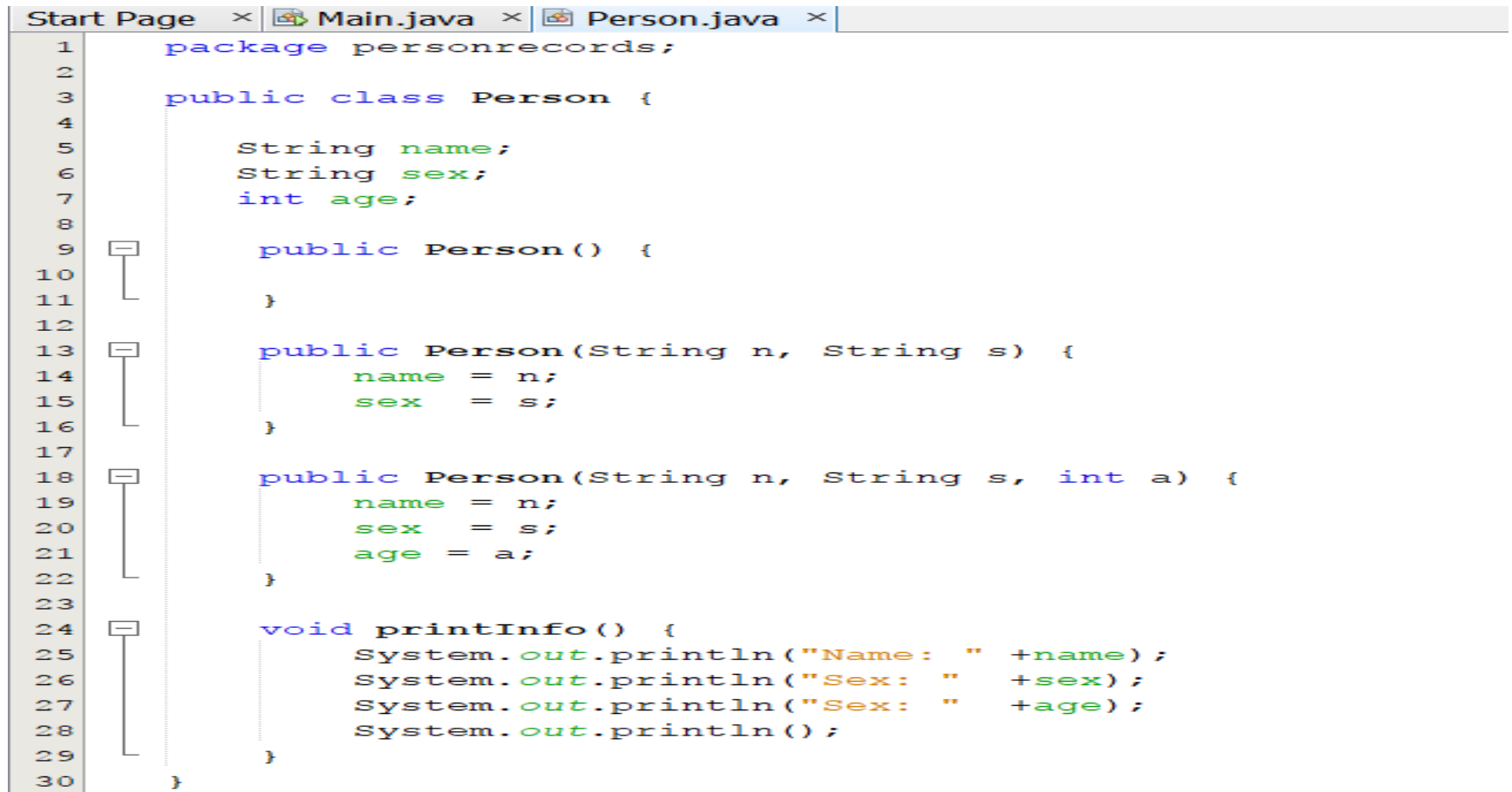
Parameterized constructor

- ▶ Parameters are the local variables of a method.
- ▶ Parameters are declared in a comma-separated parameter list.
- ▶ Each parameter must specify a type followed by a variable name.
- ▶ Multiple parameters are separated by commas.
- ▶ Each variable must be consistent with the type of the corresponding parameter.
- ▶ If the variable type does not match the parameter type, java will show an error message.

Constructor Overloading in Java

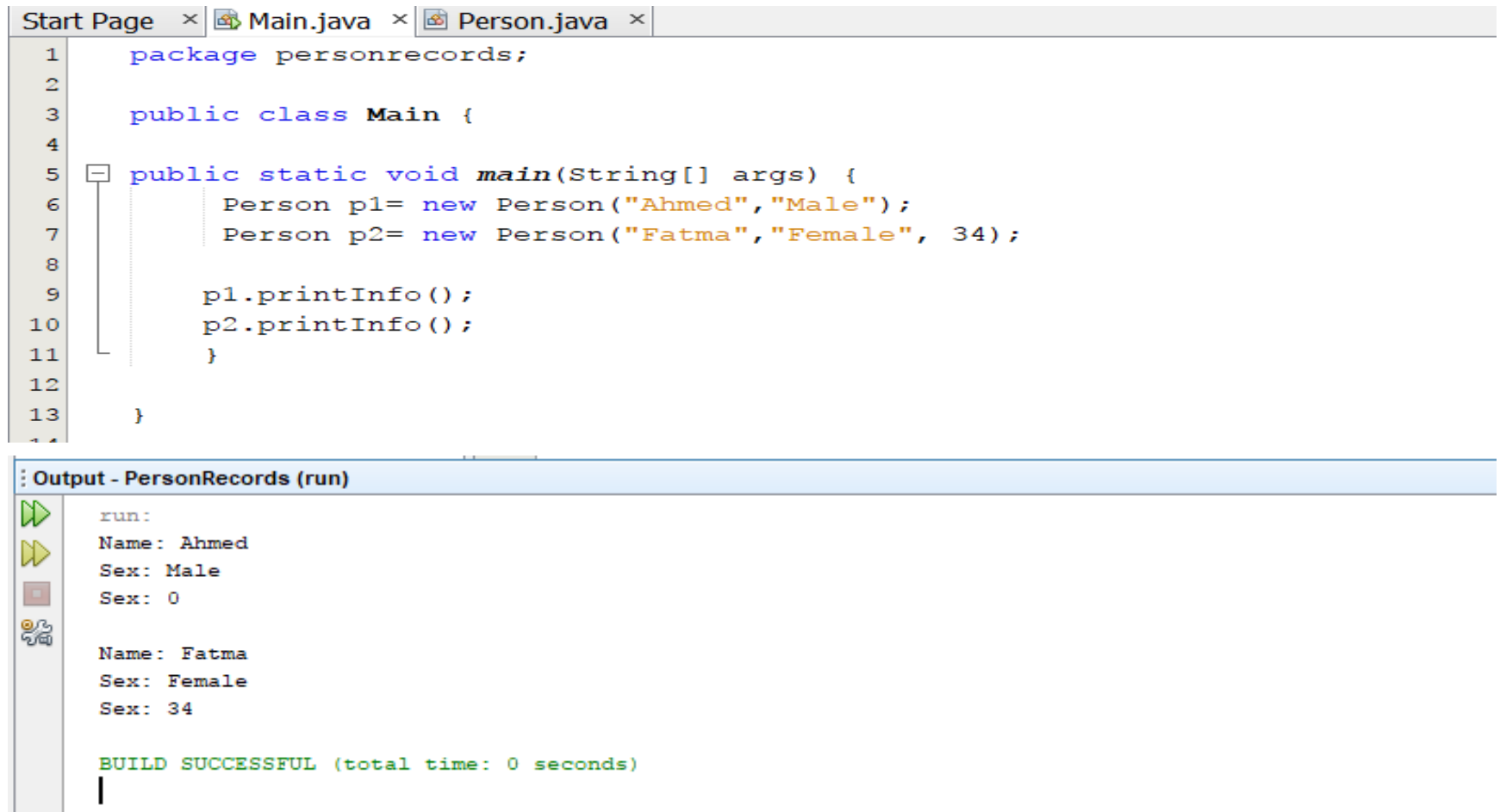
- ▶ Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- ▶ The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example



```
Start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Person {
4
5     String name;
6     String sex;
7     int age;
8
9     public Person() {
10
11     }
12
13     public Person(String n, String s) {
14         name = n;
15         sex = s;
16     }
17
18     public Person(String n, String s, int a) {
19         name = n;
20         sex = s;
21         age = a;
22     }
23
24     void printInfo() {
25         System.out.println("Name: " + name);
26         System.out.println("Sex: " + sex);
27         System.out.println("Age: " + age);
28         System.out.println();
29     }
30 }
```

Example



The screenshot shows an IDE with two tabs: 'Main.java' and 'Person.java'. The 'Main.java' tab is active, displaying the following code:

```
1 package personrecords;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Person p1= new Person("Ahmed", "Male");
7         Person p2= new Person("Fatma", "Female", 34);
8
9         p1.printInfo();
10        p2.printInfo();
11    }
12
13 }
```

Below the code editor is the 'Output - PersonRecords (run)' window. It shows the output of the program:

```
run:
Name: Ahmed
Sex: Male
Sex: 0

Name: Fatma
Sex: Female
Sex: 34

BUILD SUCCESSFUL (total time: 0 seconds)
```

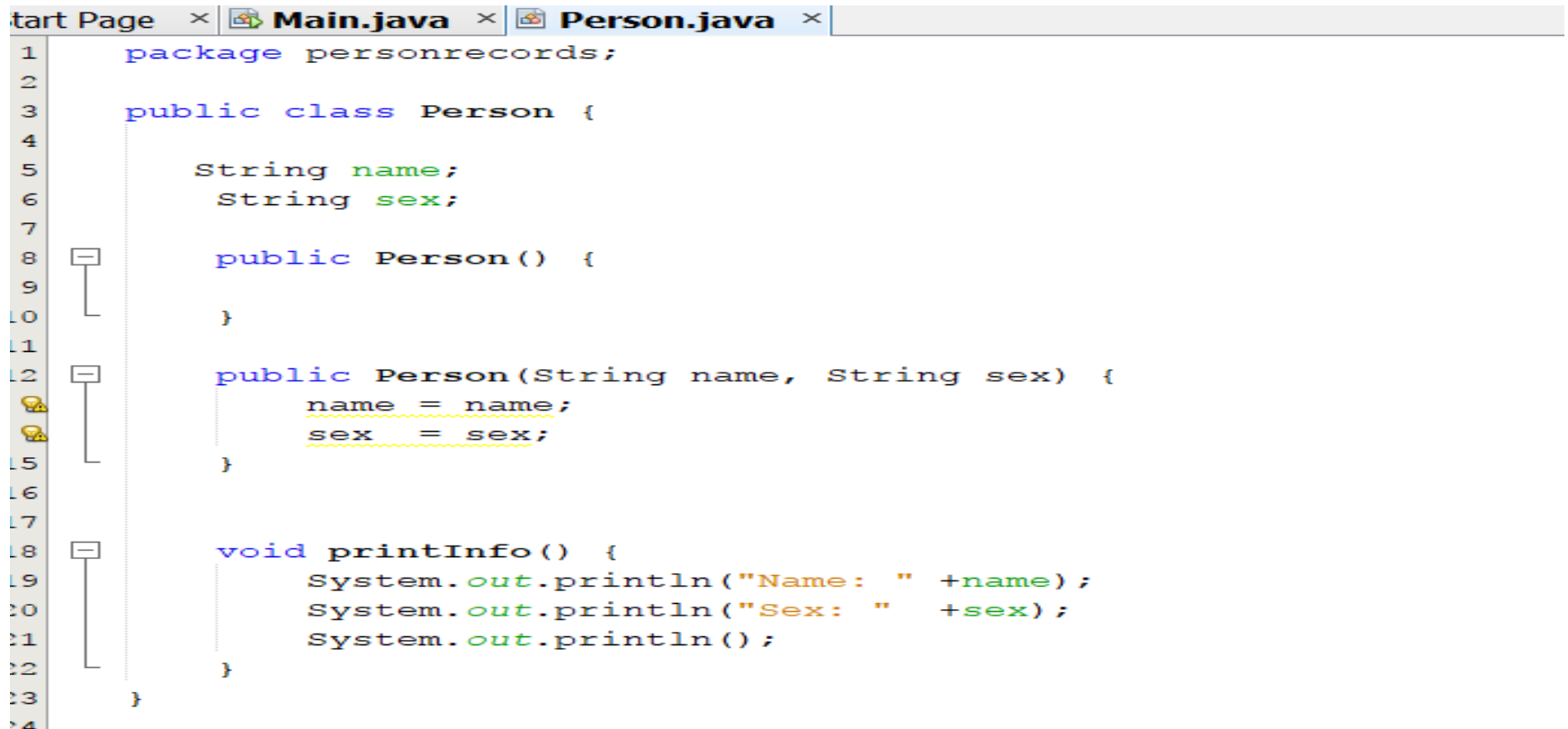
this keyword

this keyword

- ▶ There can be a lot of usage of java **this** keyword.
- ▶ In this lecture, we will look at only one usage of **this** keyword (The use of the **this** keyword to refer current class instance variable).
- ▶ The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, **this** keyword resolves the problem of ambiguity.

Understanding the problem without **this** keyword

Let's understand the problem if we don't use this keyword by the example given below:



```
1 package personrecords;
2
3 public class Person {
4
5     String name;
6     String sex;
7
8     public Person() {
9
10    }
11
12    public Person(String name, String sex) {
13        name = name;
14        sex = sex;
15    }
16
17
18    void printInfo() {
19        System.out.println("Name: " + name);
20        System.out.println("Sex: " + sex);
21        System.out.println();
22    }
23 }
24
```

Understanding the problem without **this** keyword

```
Start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Person p1= new Person("Ahmed", "Male");
7         Person p2= new Person("Fatma", "Female");
8
9         p1.printInfo();
10        p2.printInfo();
11    }
12
13 }
14
```

```
Output - PersonRecords (run)
run:
Name: null
Sex: null

Name: null
Sex: null

BUILD SUCCESSFUL (total time: 0 seconds)
|
```

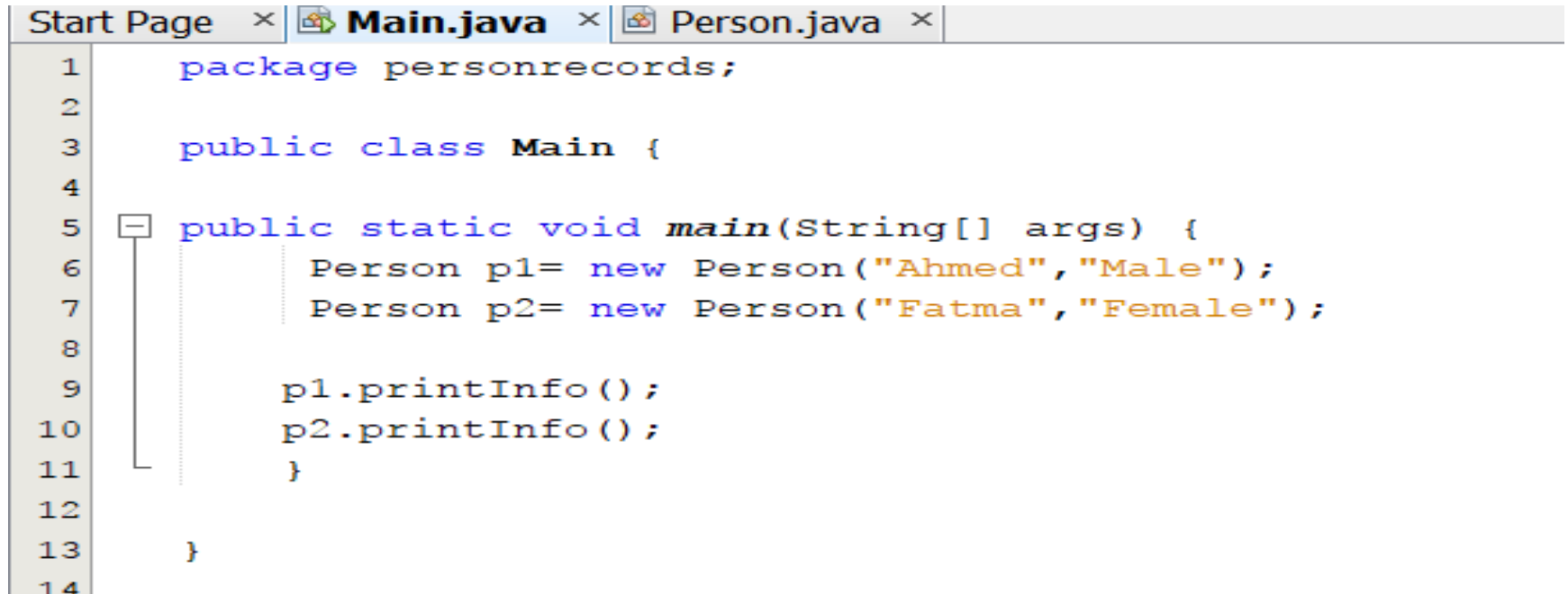
this keyword

- ▶ In the above example, parameters (formal arguments) and instance variables are same. So, we are using **this** keyword to distinguish local variable and instance variable.

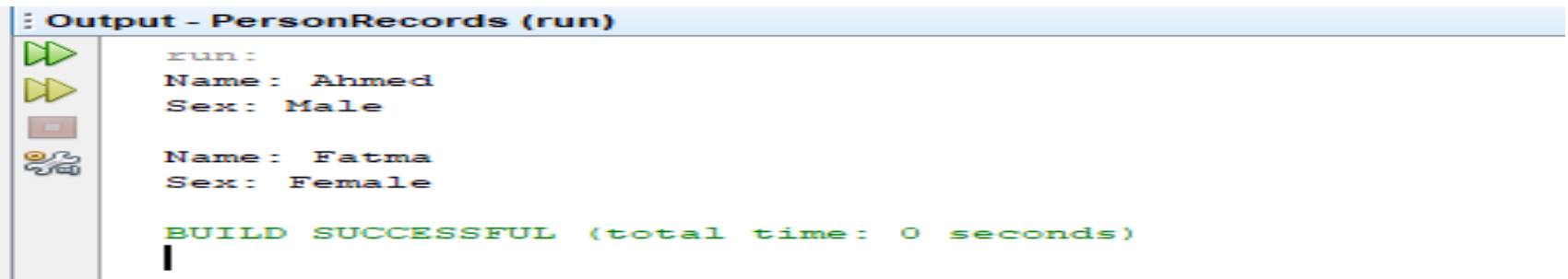
this keyword

```
start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Person {
4
5     String name;
6     String sex;
7
8     public Person() {
9
10    }
11
12    public Person(String name, String sex) {
13        this.name = name;
14        this.sex = sex;
15    }
16
17
18    void printInfo() {
19        System.out.println("Name: " + name);
20        System.out.println("Sex: " + sex);
21        System.out.println();
22    }
23 }
```

this keyword



```
Start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Main {
4
5 public static void main(String[] args) {
6     Person p1= new Person("Ahmed", "Male");
7     Person p2= new Person("Fatma", "Female");
8
9     p1.printInfo();
10    p2.printInfo();
11 }
12
13 }
14
```



```
Output - PersonRecords (run)
run:
Name: Ahmed
Sex: Male

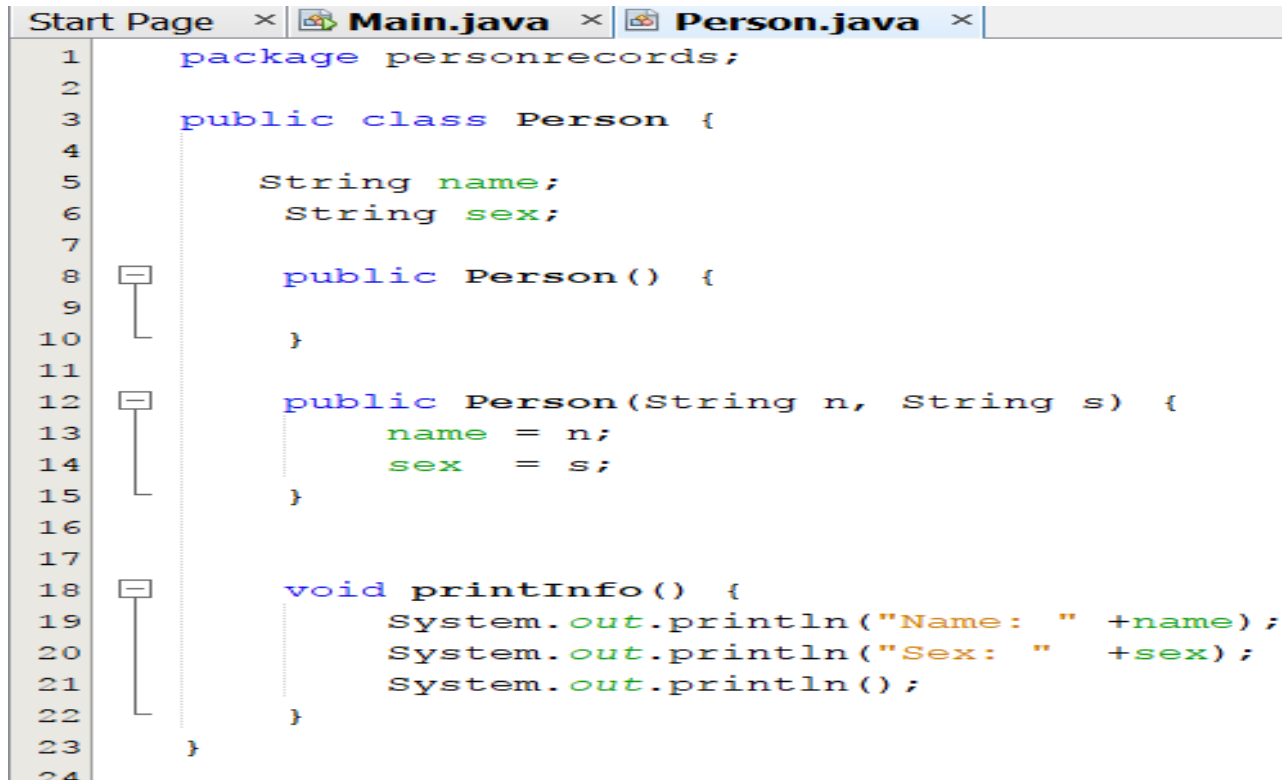
Name: Fatma
Sex: Female

BUILD SUCCESSFUL (total time: 0 seconds)
|
```

this keyword

- ▶ It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.
- ▶ If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program

this keyword



```
Start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Person {
4
5     String name;
6     String sex;
7
8     public Person() {
9
10    }
11
12    public Person(String n, String s) {
13        name = n;
14        sex = s;
15    }
16
17
18    void printInfo() {
19        System.out.println("Name: " + name);
20        System.out.println("Sex: " + sex);
21        System.out.println();
22    }
23 }
24
```

this keyword

```
Start Page x Main.java x Person.java x
1 package personrecords;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Person p1= new Person("Ahmed", "Male");
7         Person p2= new Person("Fatma", "Female");
8
9         p1.printInfo();
10        p2.printInfo();
11    }
12
13 }
14
```

```
Output - PersonRecords (run)
run:
Name: Ahmed
Sex: Male

Name: Fatma
Sex: Female

BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Assignment Two is Uploaded to the
Portal of “Learning”