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.

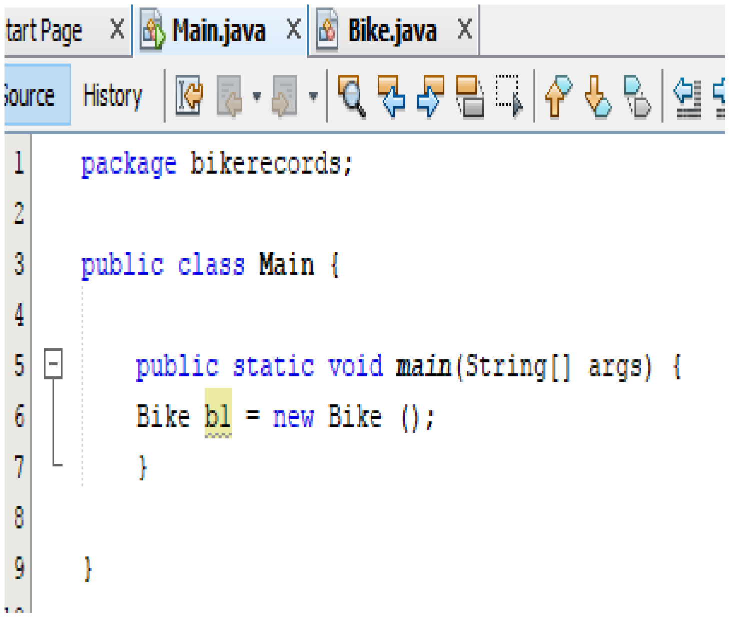
- Constructors have the following special features:

1. Constructors have the same name as the class in which they are defined, and typically should be pubic.
2. Constructors cannot have a return type: not even void.

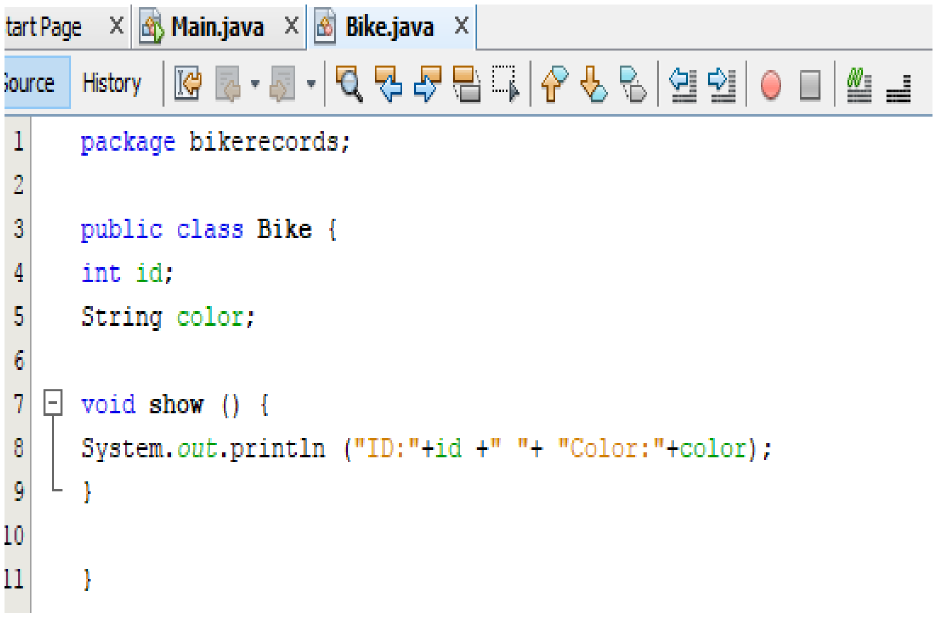
Constructor

- You must call a constructor to create each object.

Bike b1 = new $\boxed{\text{Bike () ;}}$ ← constructor



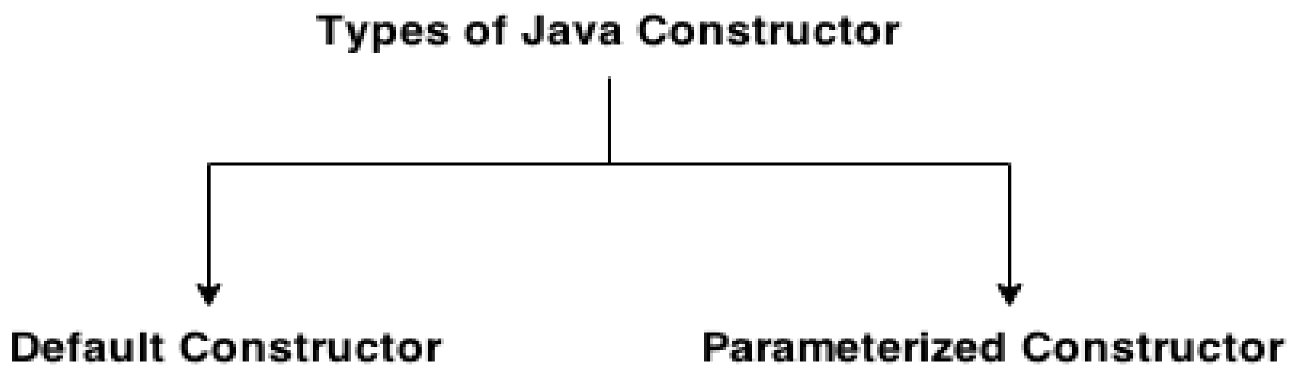
```
package bikerecords;

public class Main {

    public static void main(String[] args) {
    Bike b1 = new Bike ();
    }

}
```



```
package bikerecords;

public class Bike {
int id;
String color;

void show () {
System.out.println ("ID:"+id +" "+ "Color:"+color);
}


}
```

Constructor

- There are two types of constructors:



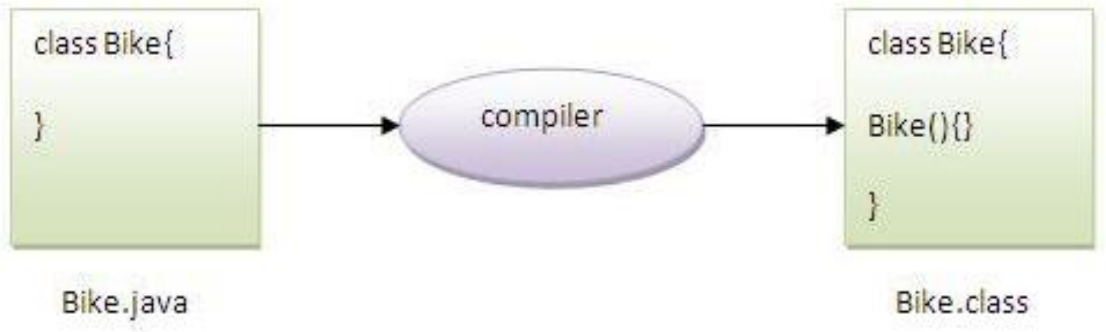
1. Default Constructor

- If a class does not define constructors, the java compiler automatically provides a default constructor with no parameters (local variables).

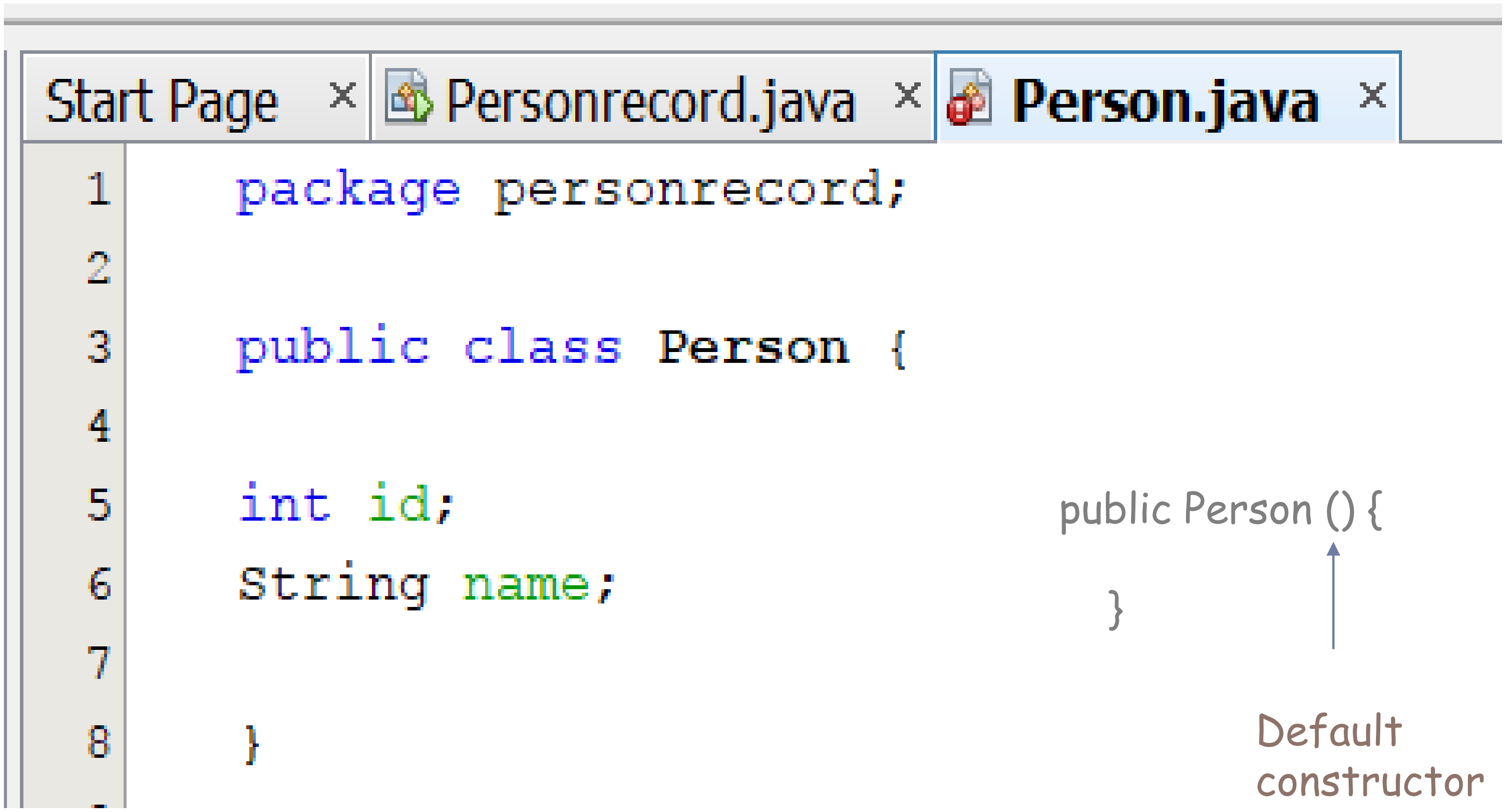
- A constructor with no parameter is called the *default constructor*.

- The default constructor provides the default values for an object.

1. Default Constructor



Example of default constructor

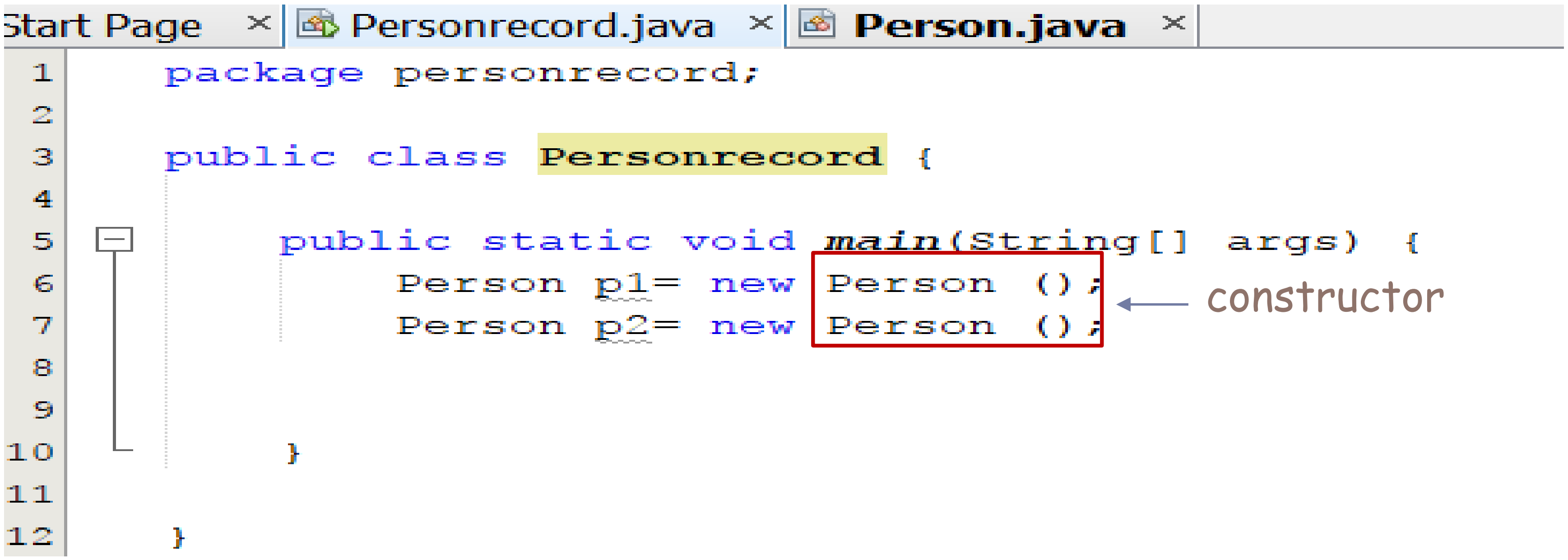


```
package personrecord;

public class Person {

int id;
String name;

}
```

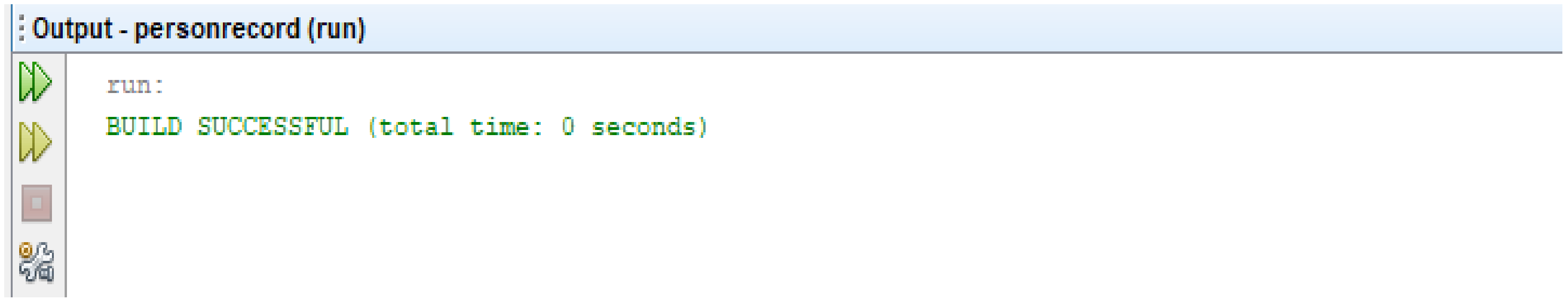
```
public Person () {

    }
```

Default constructor with no parameters

Example of default constructor



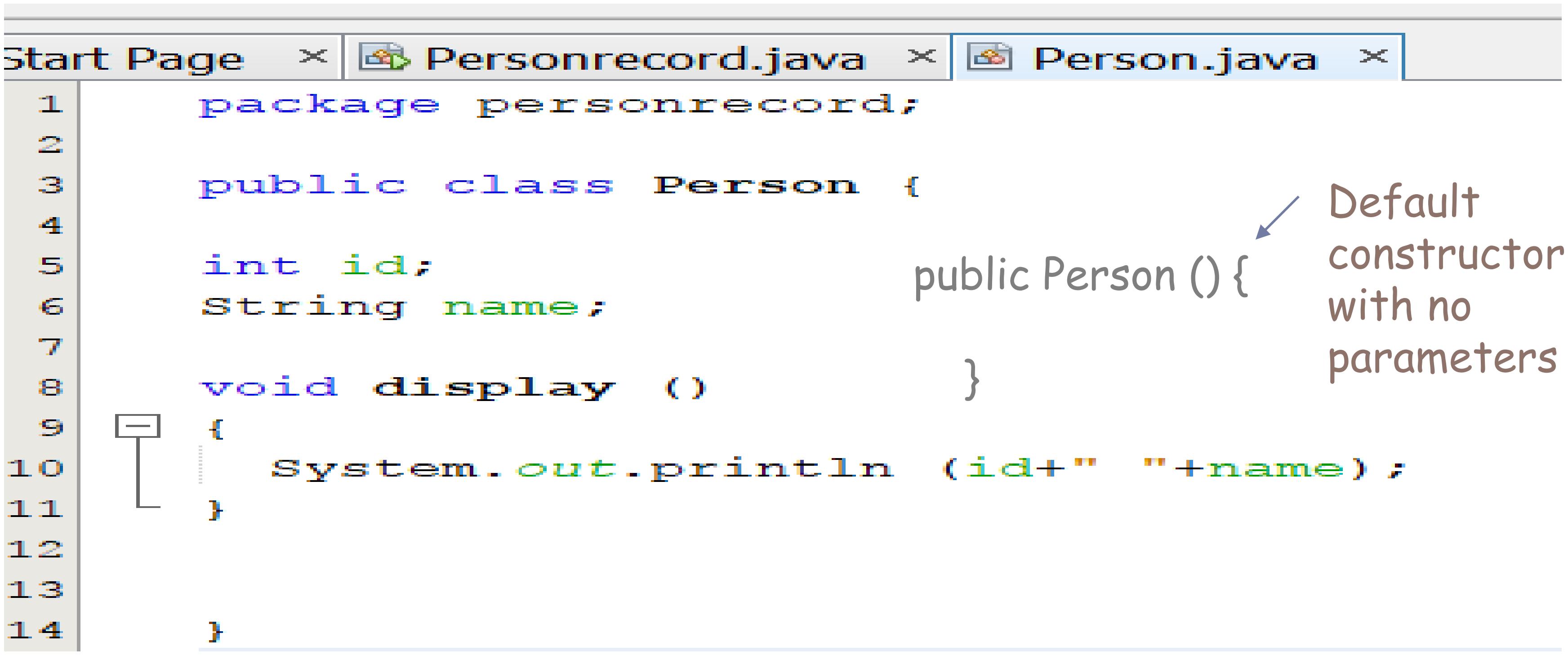
```java
package personrecord;

public class Personrecord {

    public static void main(String[] args) {
        Person p1= new Person ();
        Person p2= new Person ();


    }

}
```



```
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

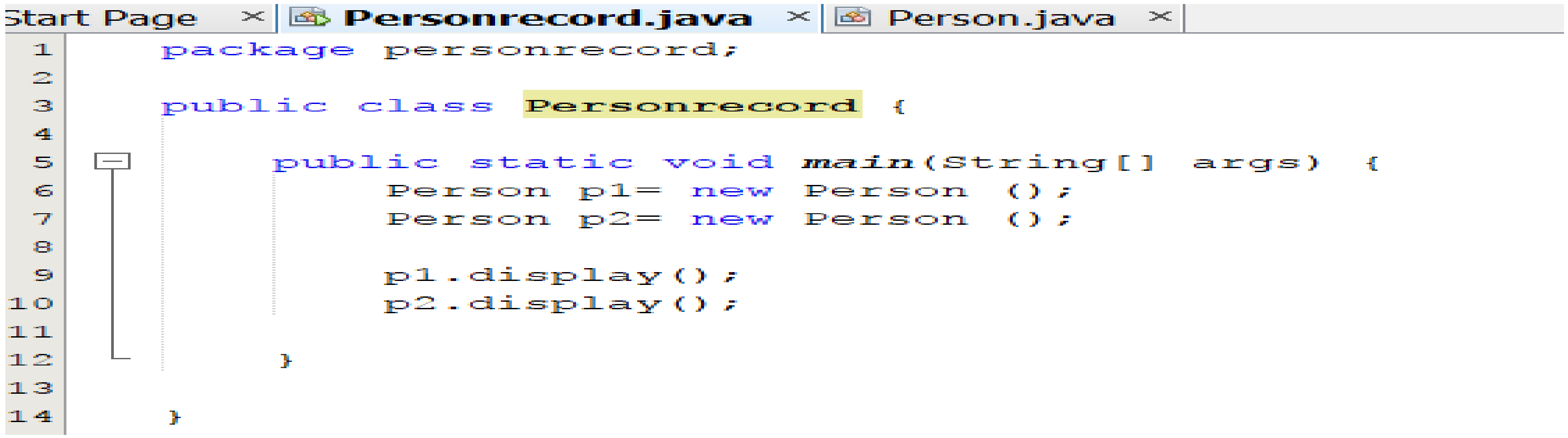


```
package personrecord;

public class Person {

int id;
String name;

void display ()
{
   System.out.println (id+" "+name);
}


}
```

```
public Person () {

}
```

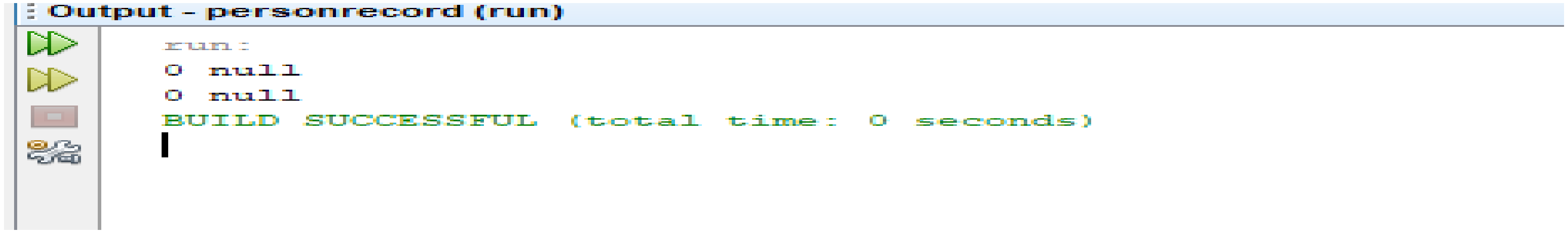
Default constructor with no parameters

Example of default constructor that displays the default values



Start Page × | Personrecord.java × | Person.java ×

```
package personrecord;

public class Personrecord {

    public static void main(String[] args) {
        Person p1= new Person ();
        Person p2= new Person ();

        p1.display();
        p2.display();

    }

}
```



Output - personrecord (run)

```
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?

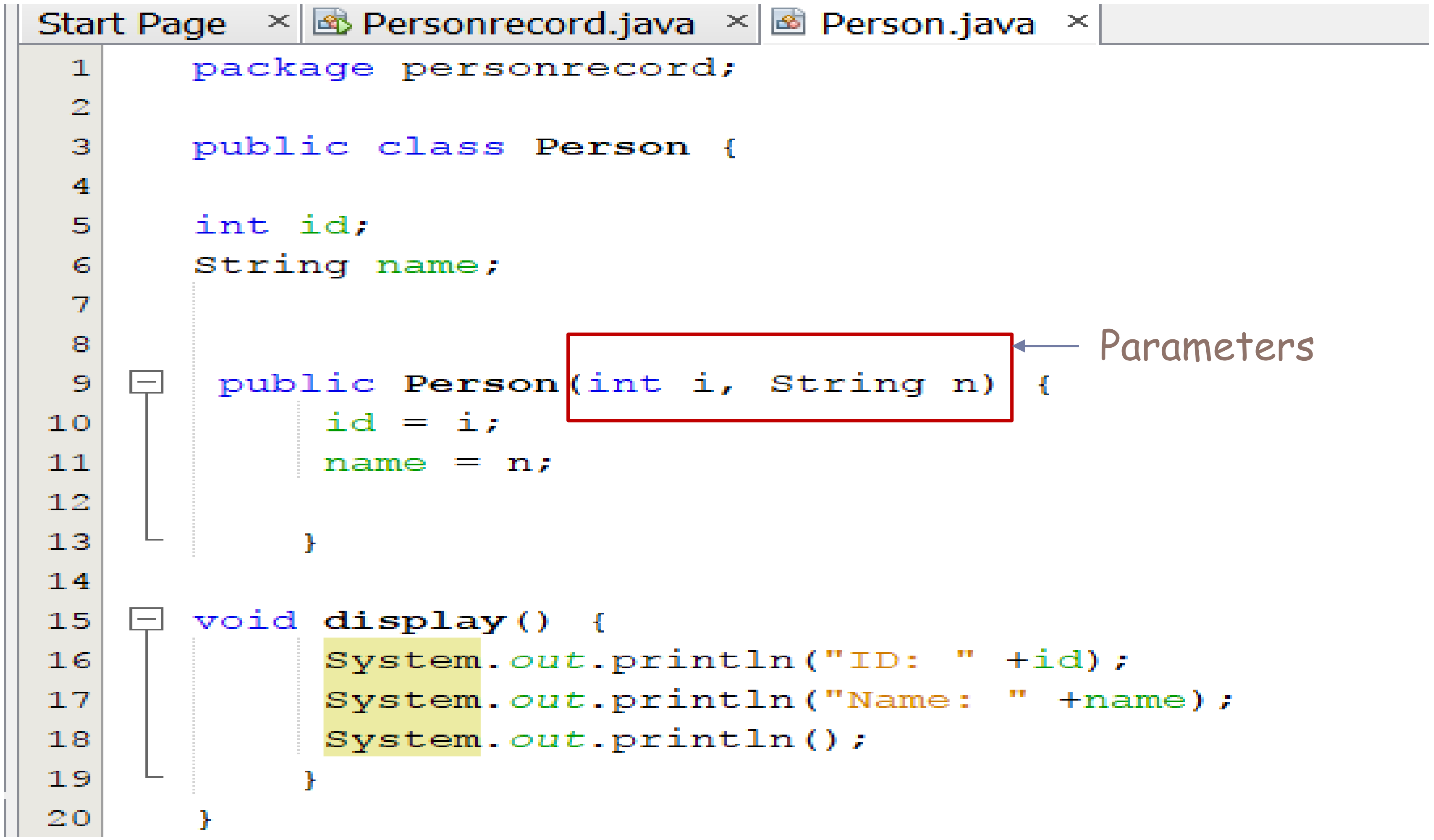
  Parameterized constructor is used to provide different values to distinct objects.

- In case you create a constructor, then java will not create a default constructor.

Example 1 of parameterized constructor

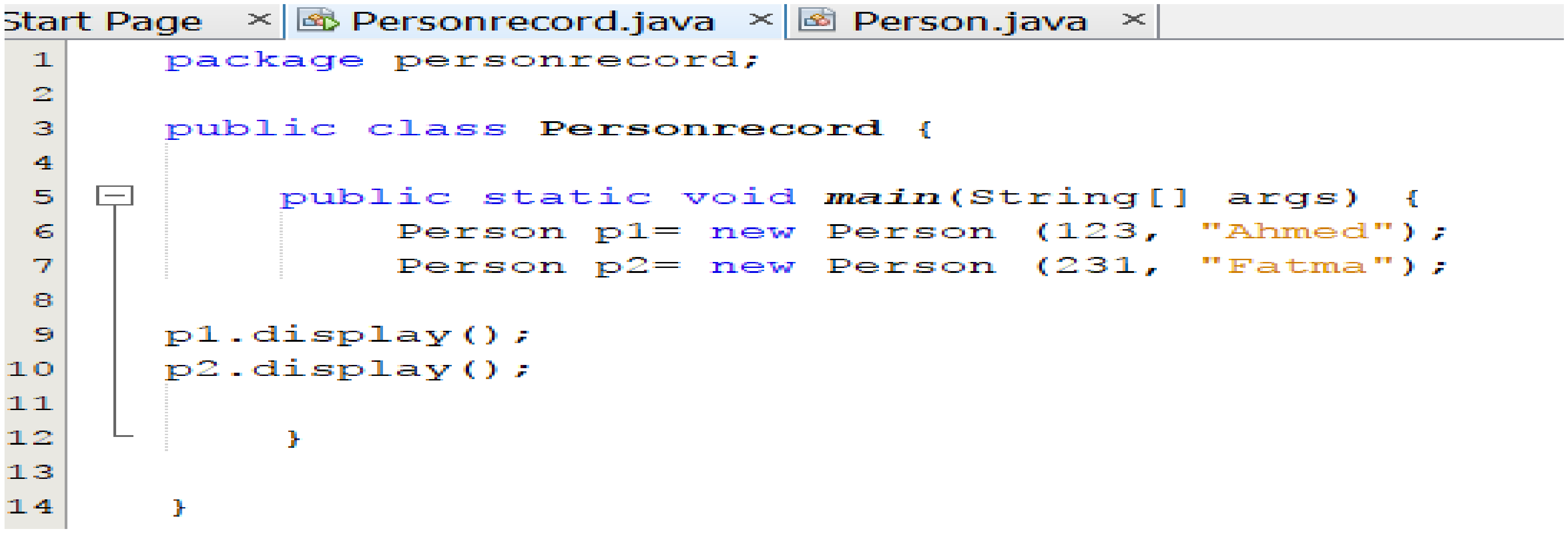
- In this example, we have created a parametrized constructor of Person class that have two parameters.
- We can have any number of parameters in the constructor.

Example 1 of parameterized constructor



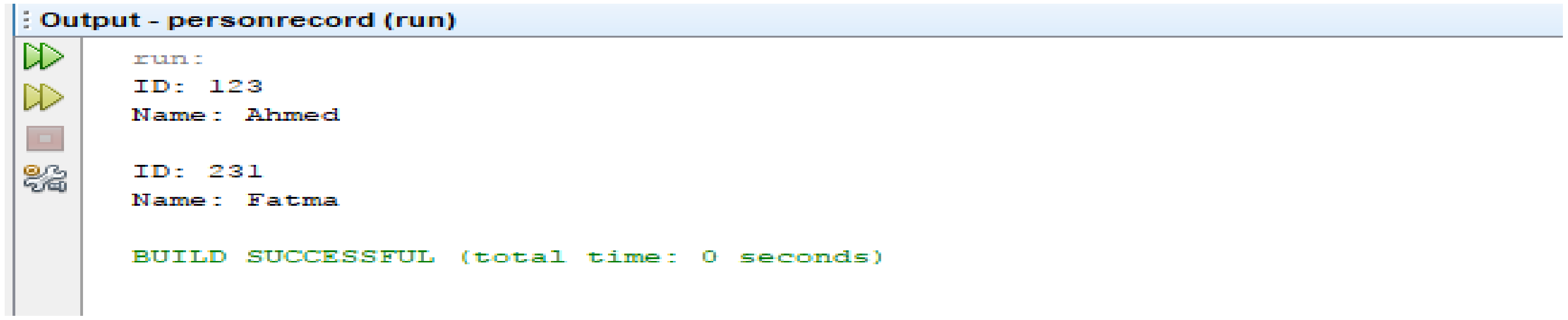
```
package personrecord;

public class Person {

int id;
String name;


 public Person(int i, String n) {
     id = i;
     name = n;

     }

void display() {
     System.out.println("ID:  " +id);
     System.out.println("Name:  " +name);
     System.out.println();
     }
}
```

Example 1 of parameterized constructor



Start Page | Personrecord.java | Person.java

```
package personrecord;

public class Personrecord {

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

p1.display();
p2.display();

    }

}
```



Output - personrecord (run)

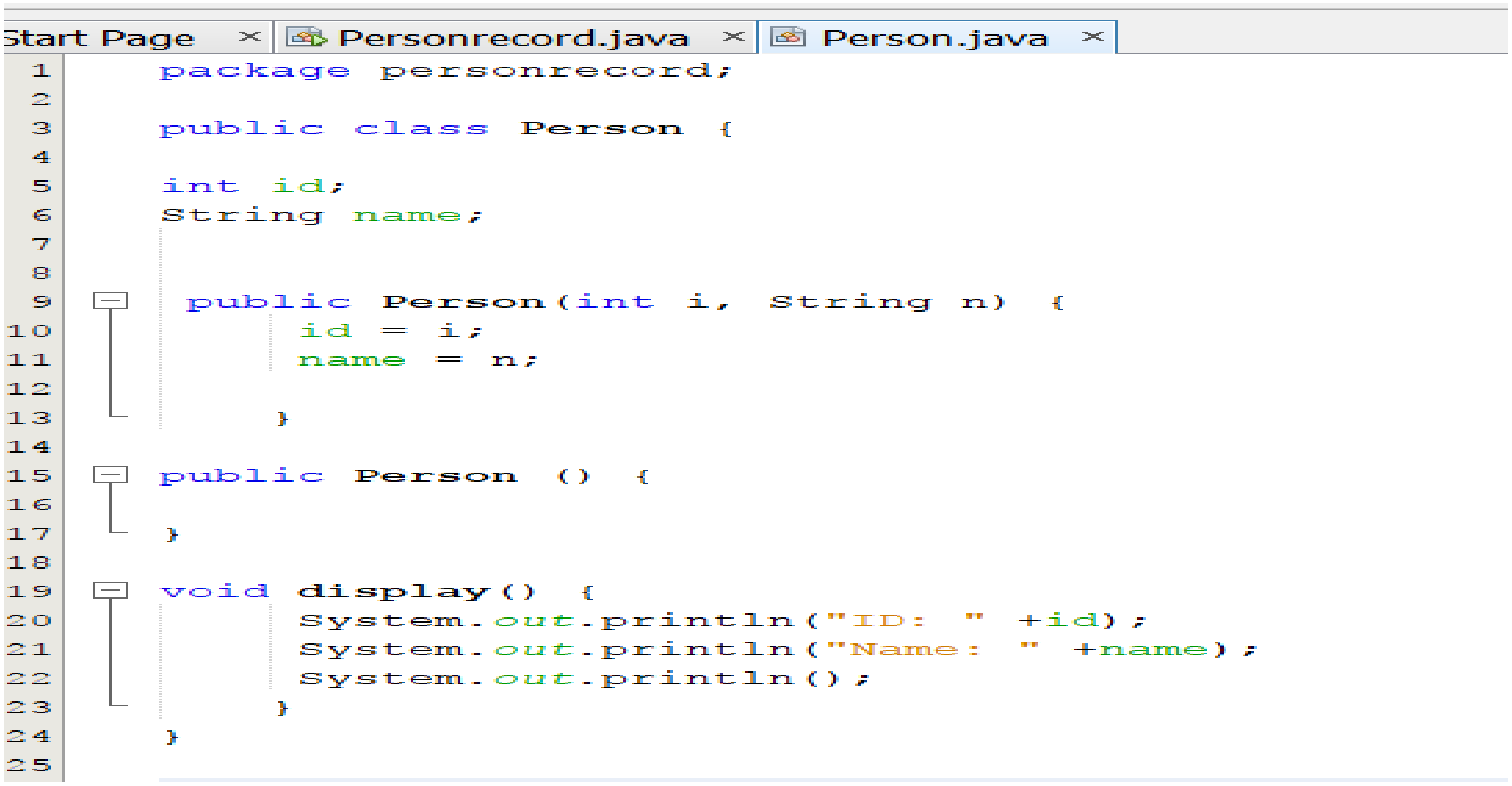
```
run:
ID: 123
Name: Ahmed

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.

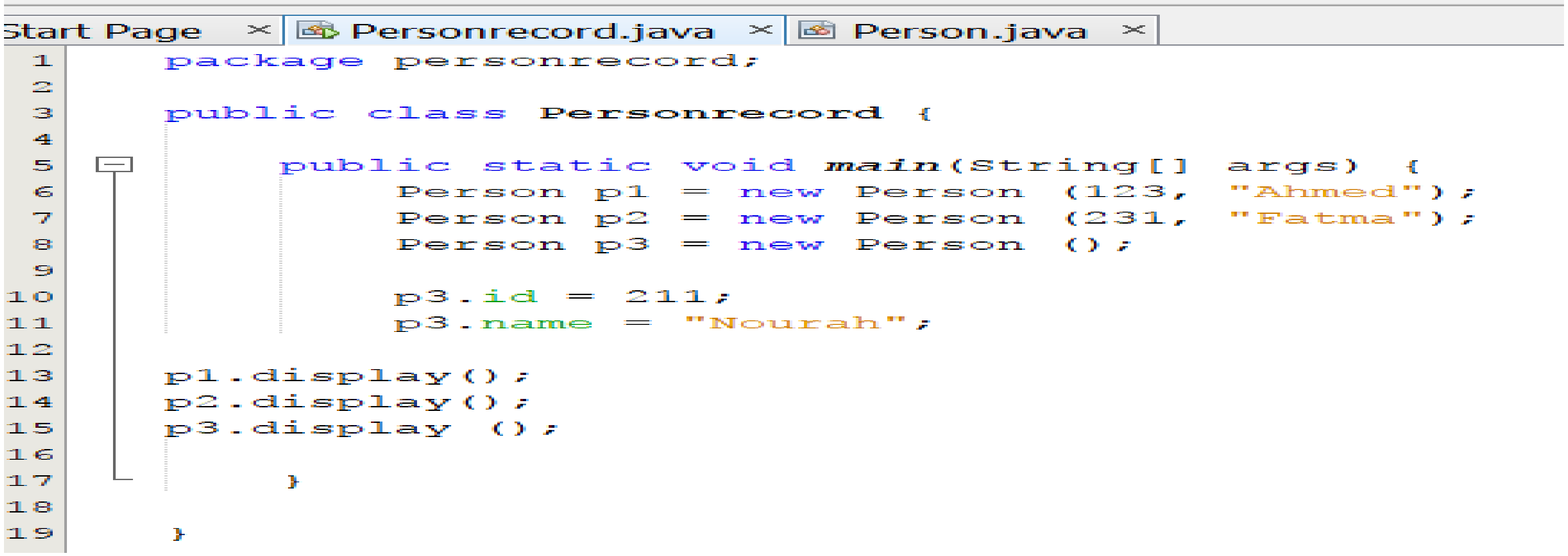
Example 2 of parameterized constructor



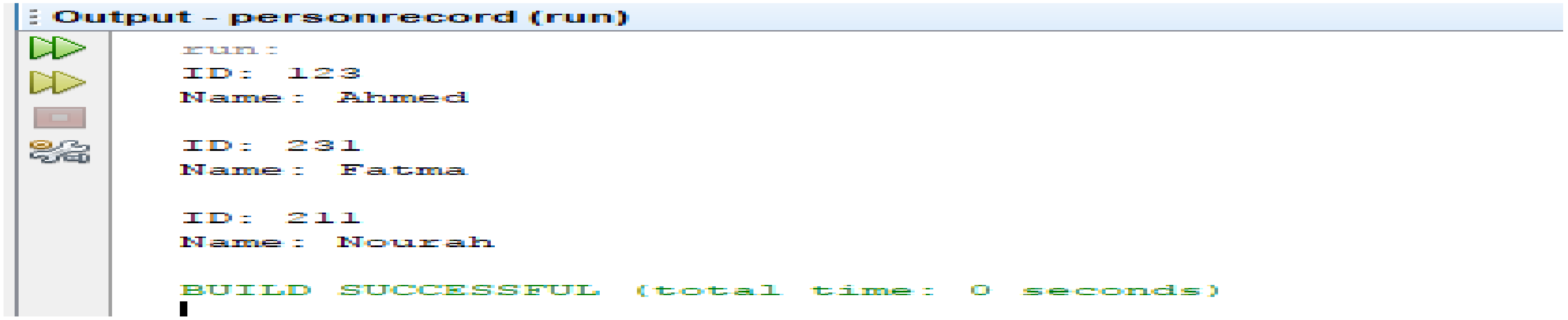
Start Page × Personrecord.java × Person.java ×

```
package personrecord;

public class Person {

int id;
String name;


 public Person(int i, String n)  {
        id = i;
        name = n;

    }

public Person ()  {

}

void display()  {
        System.out.println("ID:  " +id);
        System.out.println("Name:  " +name);
        System.out.println();
    }
}

```

Example 2 of parameterized constructor

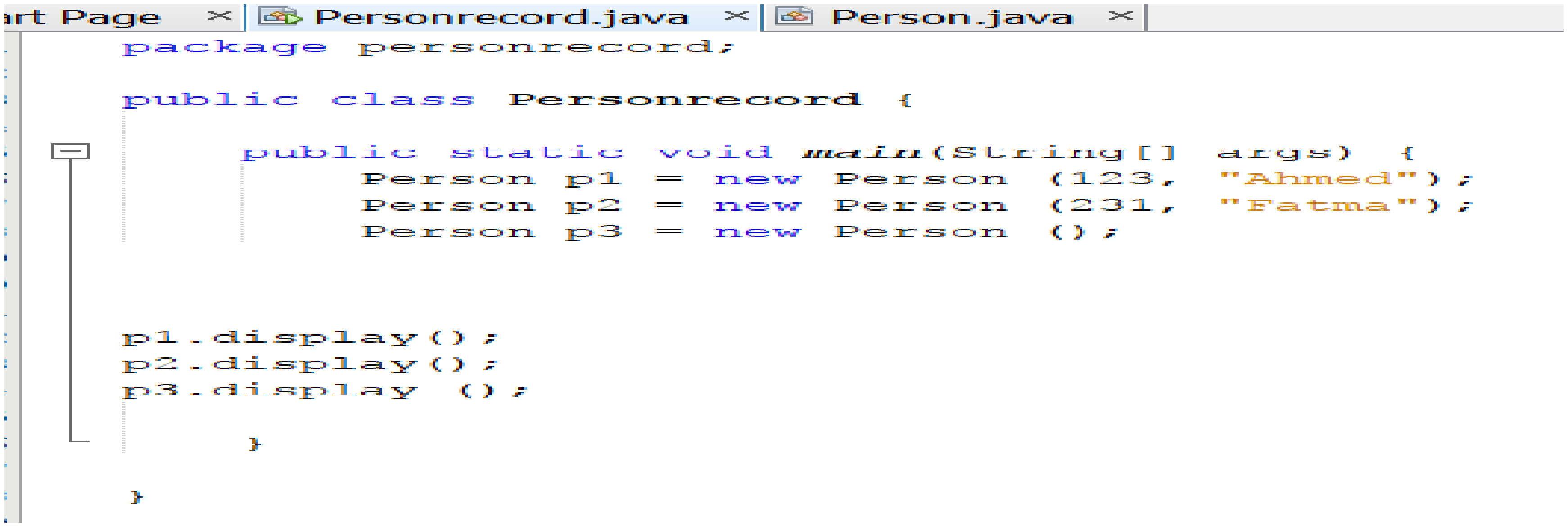

Start Page | Personrecord.java | Person.java

```java
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 ();

            p3.id = 211;
            p3.name = "Nourah";

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

        }

}
```

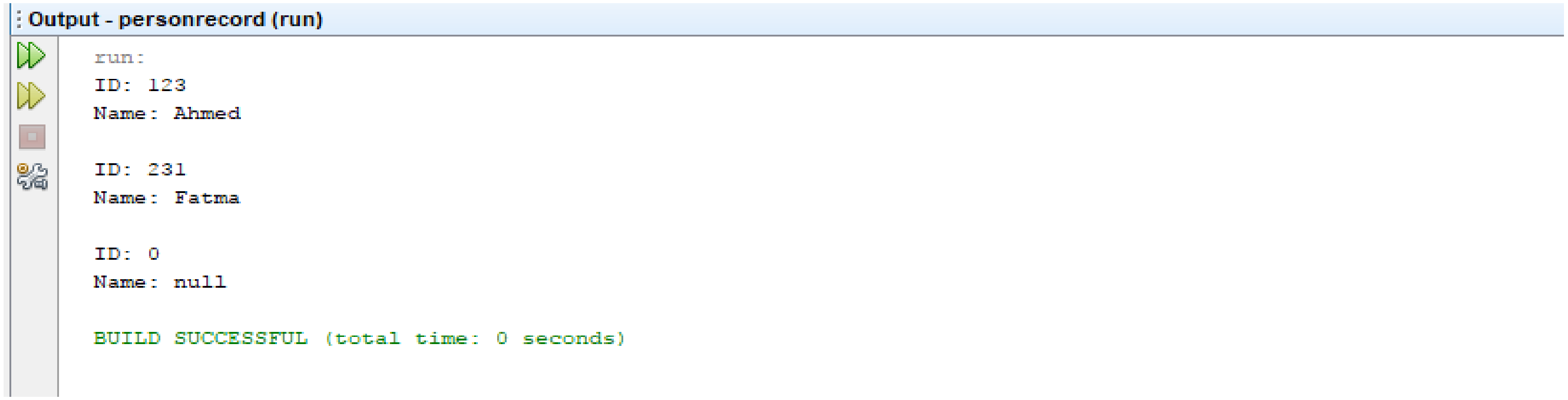

Output - personrecord (run)

```
run:
ID: 123
Name: Ahmed

ID: 231
Name: Fatma

ID: 211
Name: Nourah

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

Example 2 of parameterized constructor



```
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 ();

    }

}
```



```
run:
ID: 123
Name: Ahmed

ID: 231
Name: Fatma

ID: 0
Name: null

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

Parameterized constructor

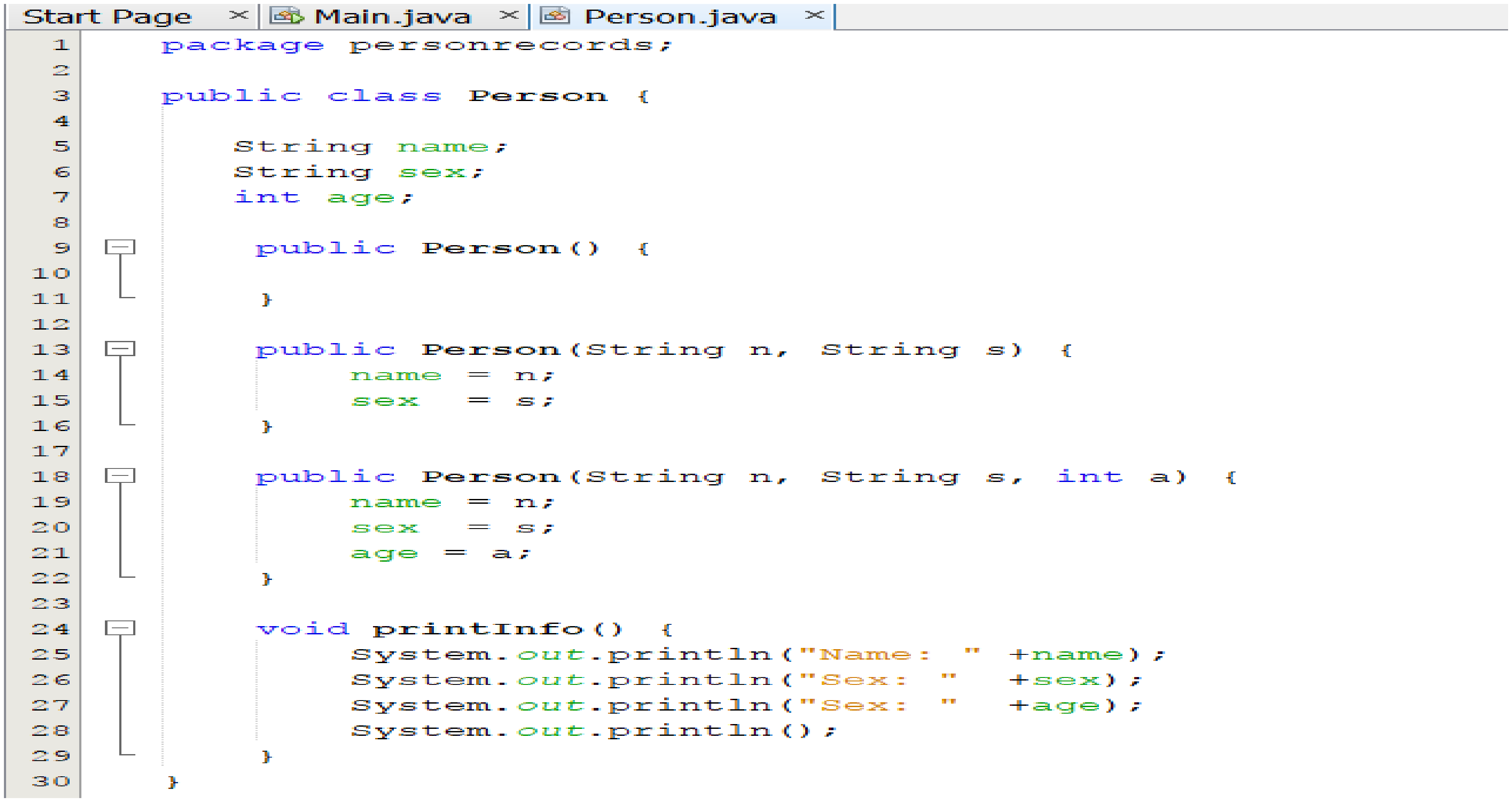
- Parameters are the local variables of a method.
- Parameters are declared in a comma-separated parameter list.
- Each parameter must specify a type followed by a variable name.
- Multiple parameters are separated by commas.
- Each variable must be consistent with the type of the corresponding parameter.
- 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.

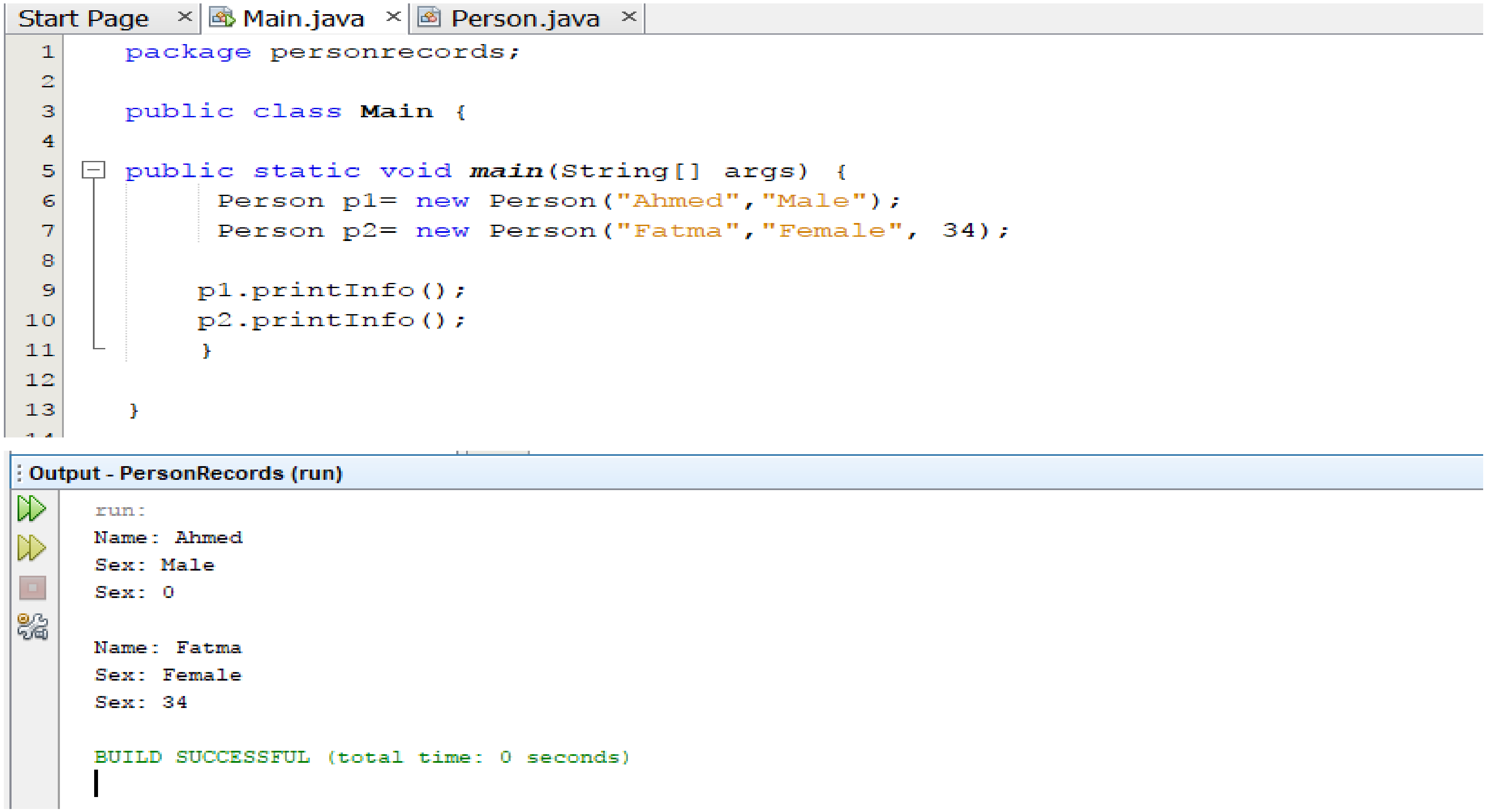
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example


Start Page   Main.java   Person.java

```java
package personrecords;

public class Person {

    String name;
    String sex;
    int age;

     public Person()  {

     }

     public Person(String n, String s)  {
         name = n;
         sex  = s;
     }

     public Person(String n, String s, int a)  {
         name = n;
         sex  = s;
         age = a;
     }

     void printInfo()  {
         System.out.println("Name:  " +name);
         System.out.println("Sex:  "  +sex);
         System.out.println("Sex:  "  +age);
         System.out.println();
     }
}
```

Example


Start Page × Main.java × Person.java ×

```
package personrecords;

public class Main {

public static void main(String[] args) {
        Person p1= new Person("Ahmed","Male");
        Person p2= new Person("Fatma","Female", 34);

    p1.printInfo();
    p2.printInfo();
    }

}
```

Output - PersonRecords (run)

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

Name: Fatma
Sex: Female
Sex: 34

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

this keyword

this keyword

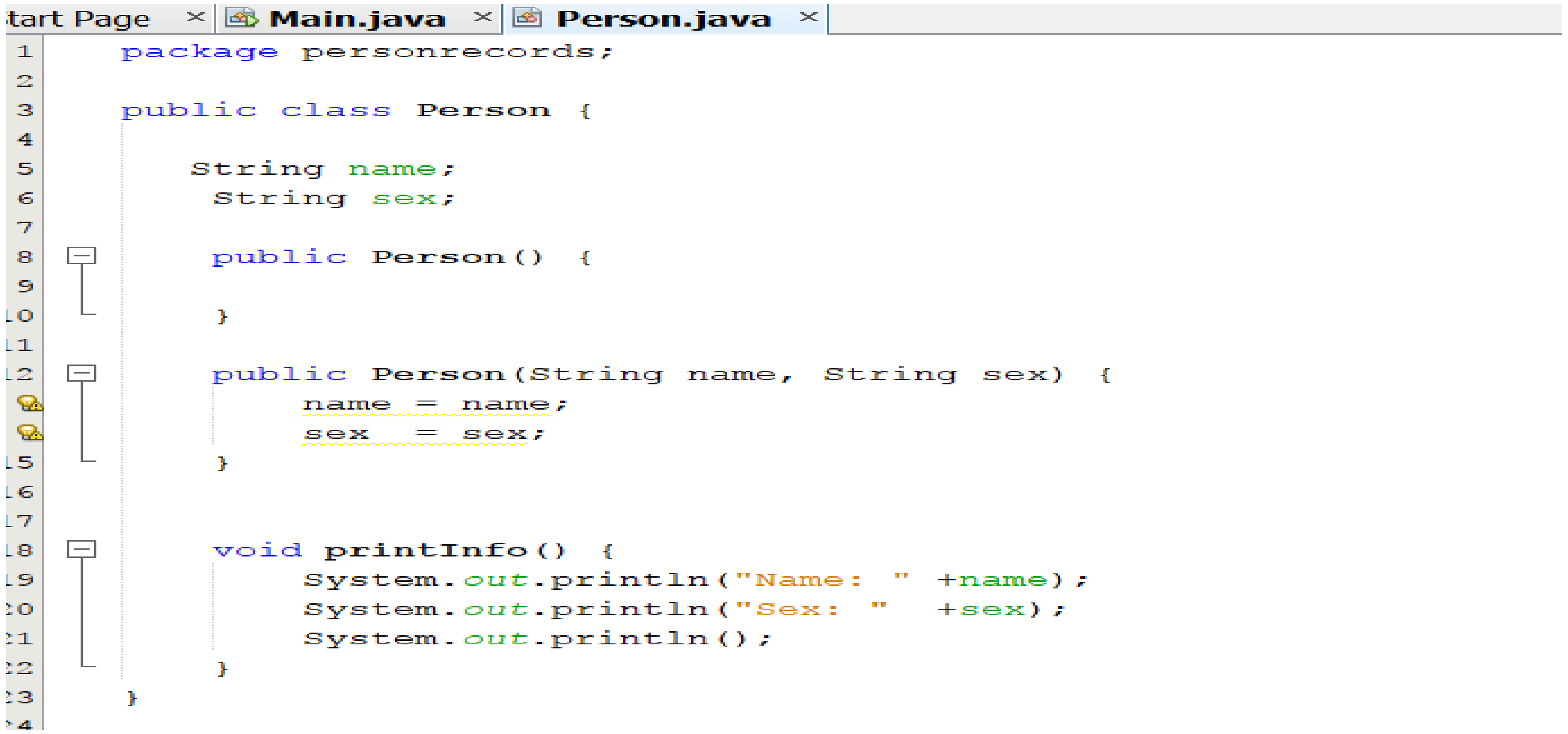
- There can be a lot of usage of java **this** keyword.

- 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

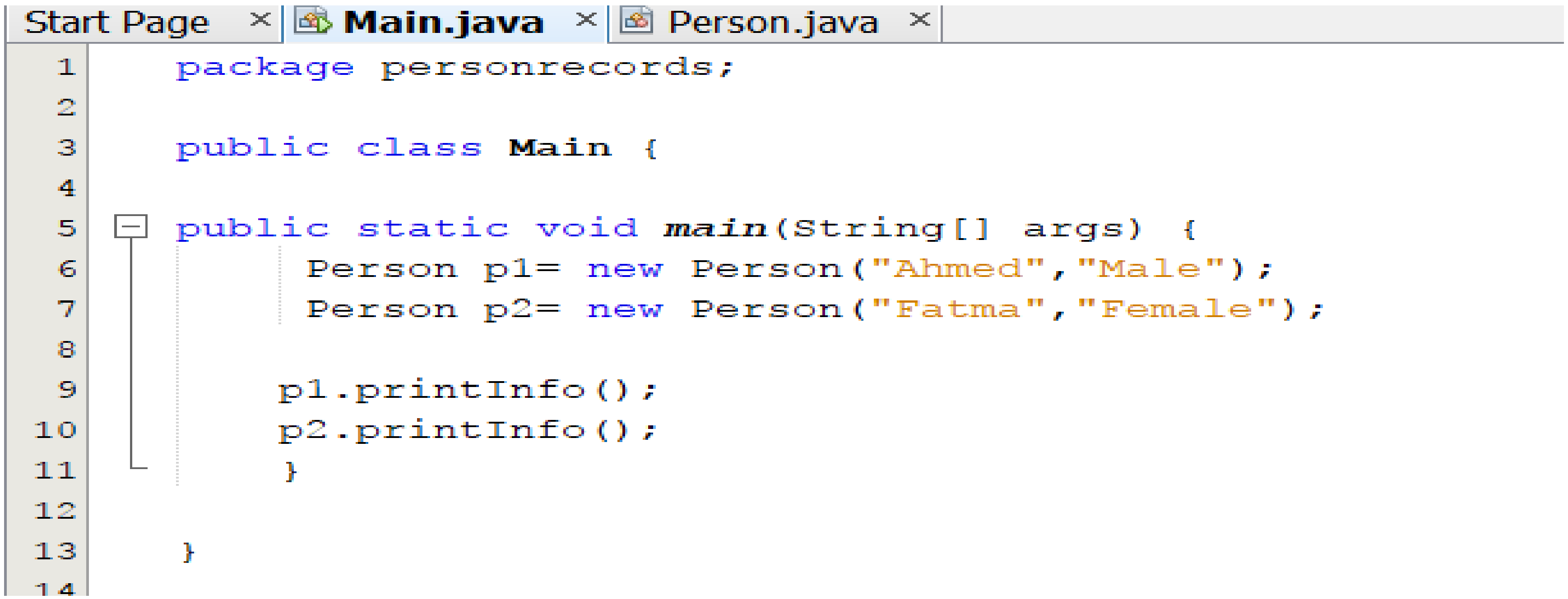
Let's understand the problem if we don't use this keyword by the example given below:



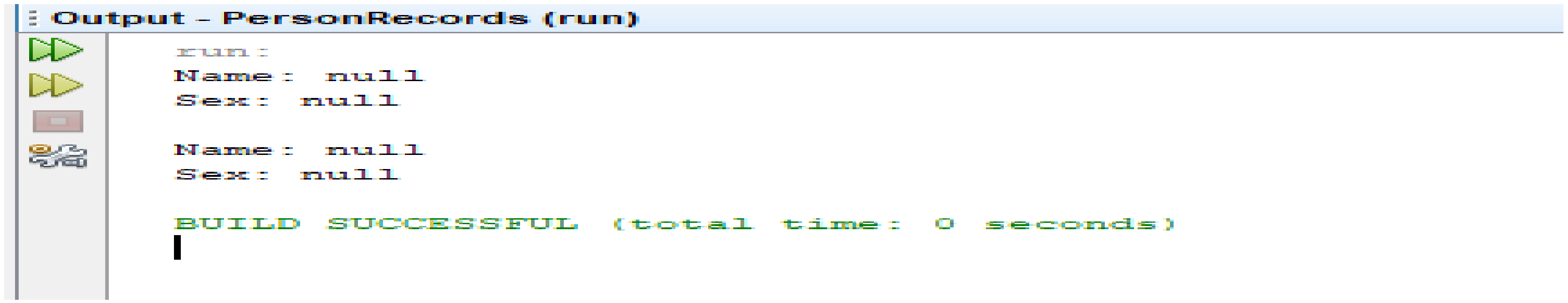
tart Page × | Main.java × | Person.java ×

```
    package personrecords;

    public class Person {

        String name;
         String sex;

        public Person()  {

        }

        public Person(String name, String sex)  {
            name  = name;
            sex   = sex;
        }


        void printInfo()  {
            System.out.println("Name:  " +name);
            System.out.println("Sex:  "  +sex);
            System.out.println();
        }
    }
```

Understanding the problem without **this** keyword



```java
package personrecords;

public class Main {

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

      p1.printInfo();
      p2.printInfo();
      }

}
```

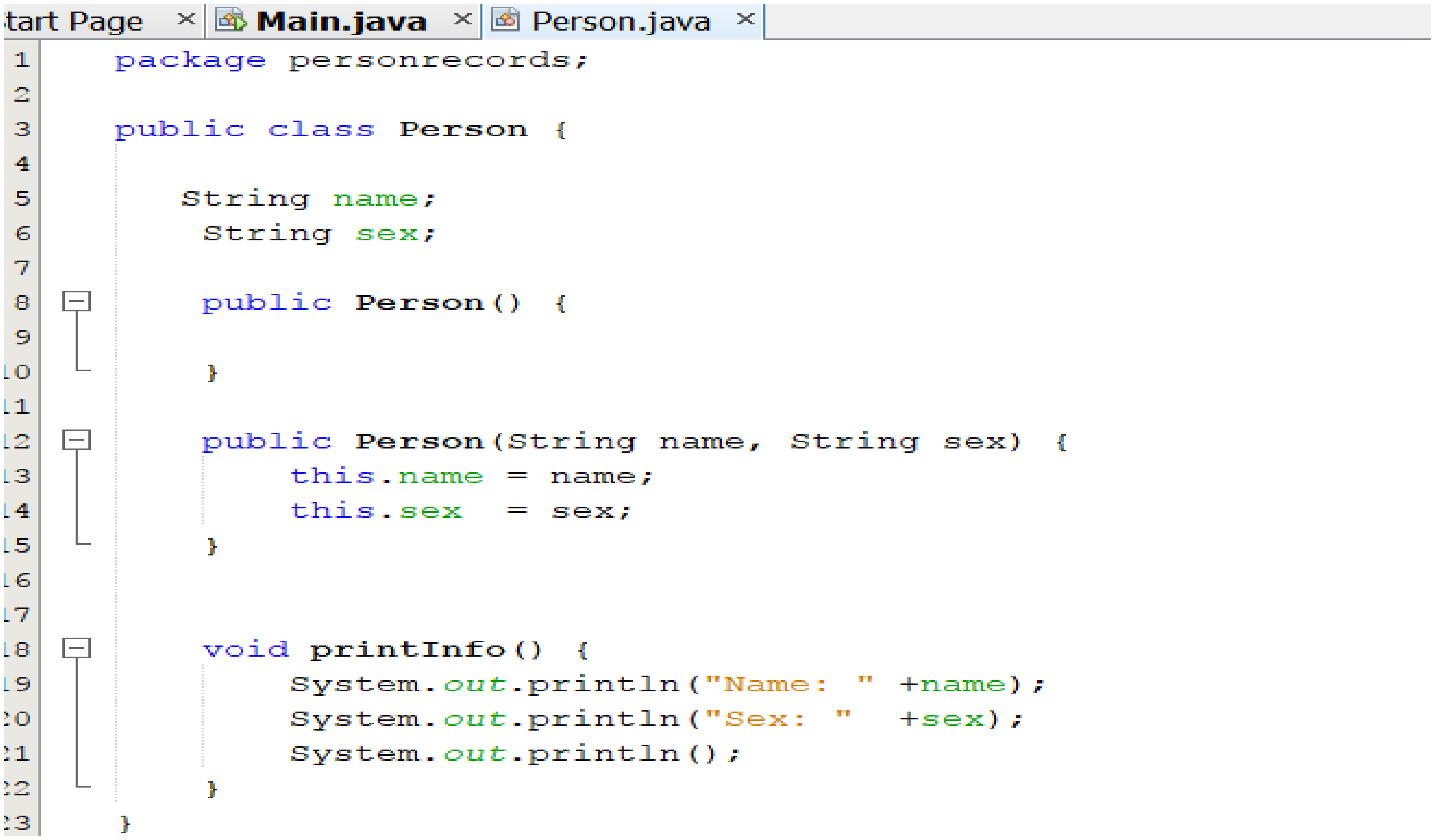


```
run:
Name: null
Sex: null

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.

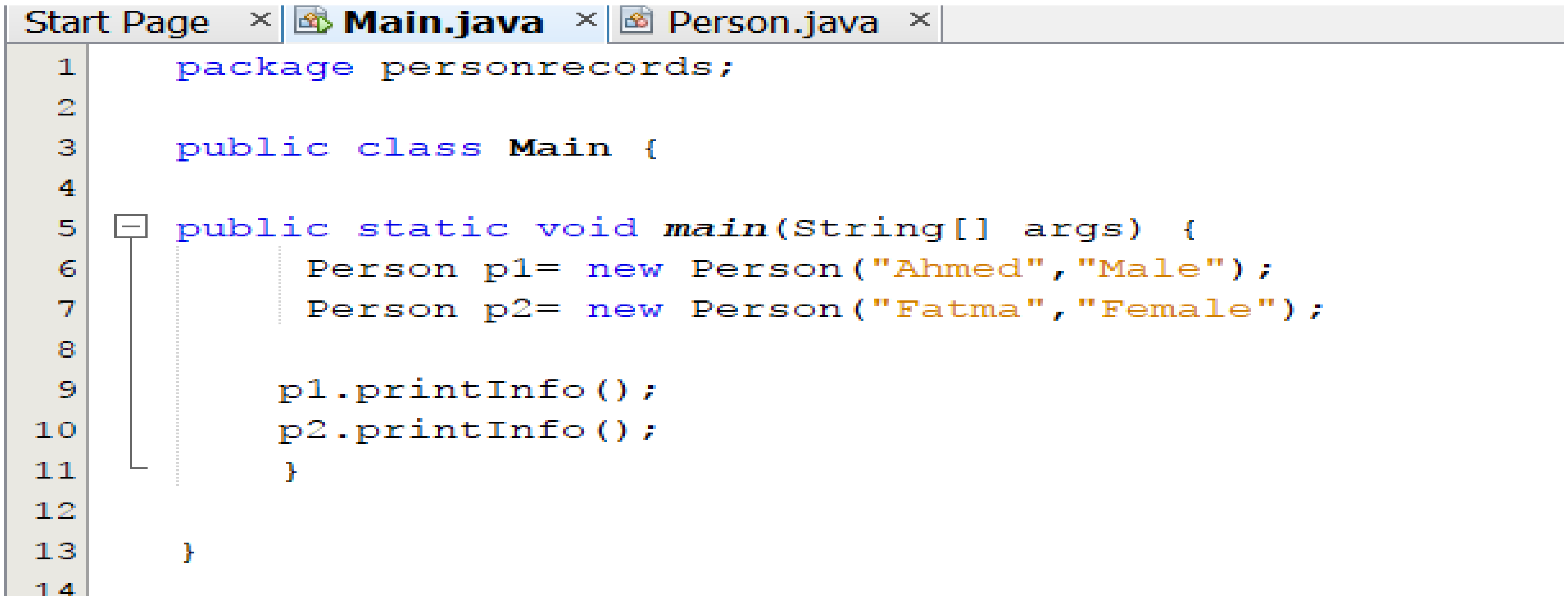
this keyword



tart Page ✕ Main.java ✕ Person.java ✕

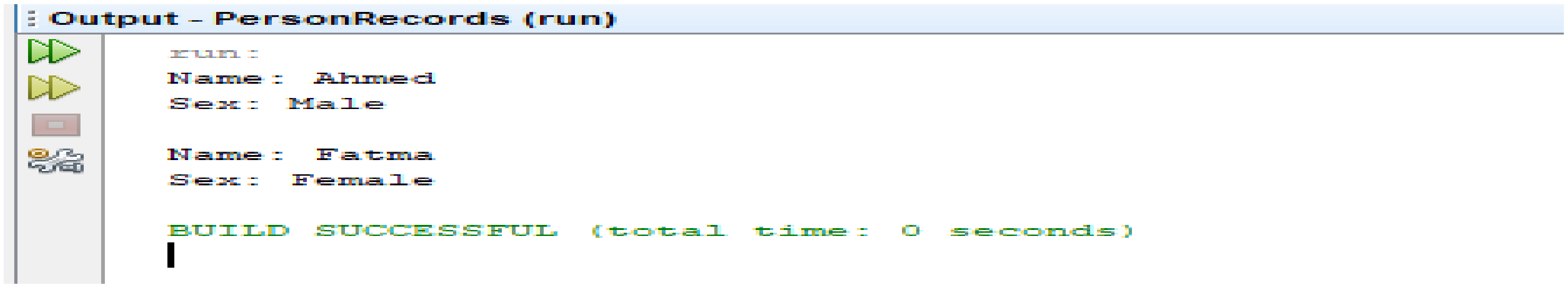
```
    package personrecords;

    public class Person {

        String name;
         String sex;

         public Person()  {

         }

         public Person(String name, String sex)  {
             this.name  = name;
             this.sex   = sex;
         }


         void printInfo()  {
             System.out.println("Name:  " +name);
             System.out.println("Sex:  "  +sex);
             System.out.println();
         }
}
```

this keyword



Start Page × | Main.java × | Person.java ×

```java
package personrecords;

public class Main {

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

      p1.printInfo();
      p2.printInfo();
      }

}
```



Output - PersonRecords (run)

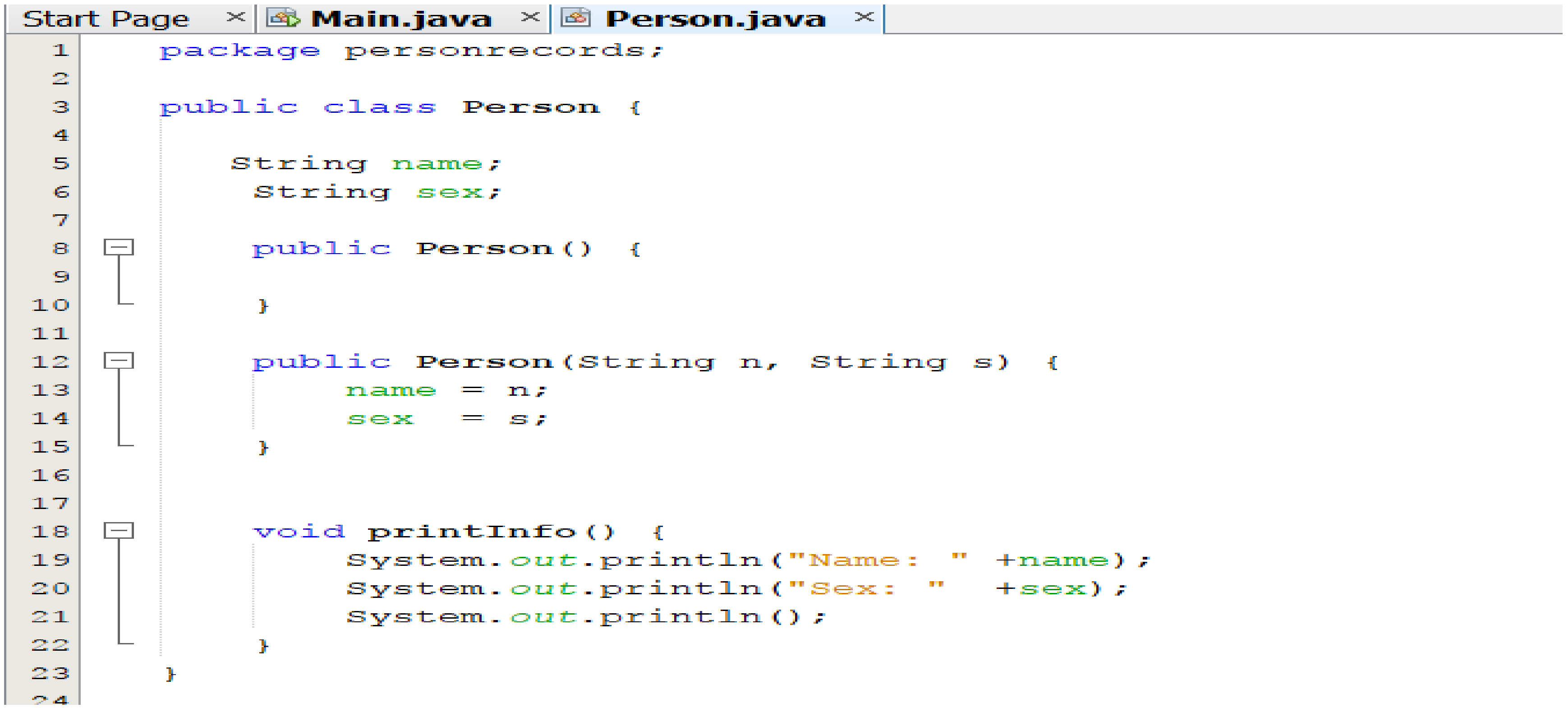
```
run:
Name:  Ahmed
Sex:  Male

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.

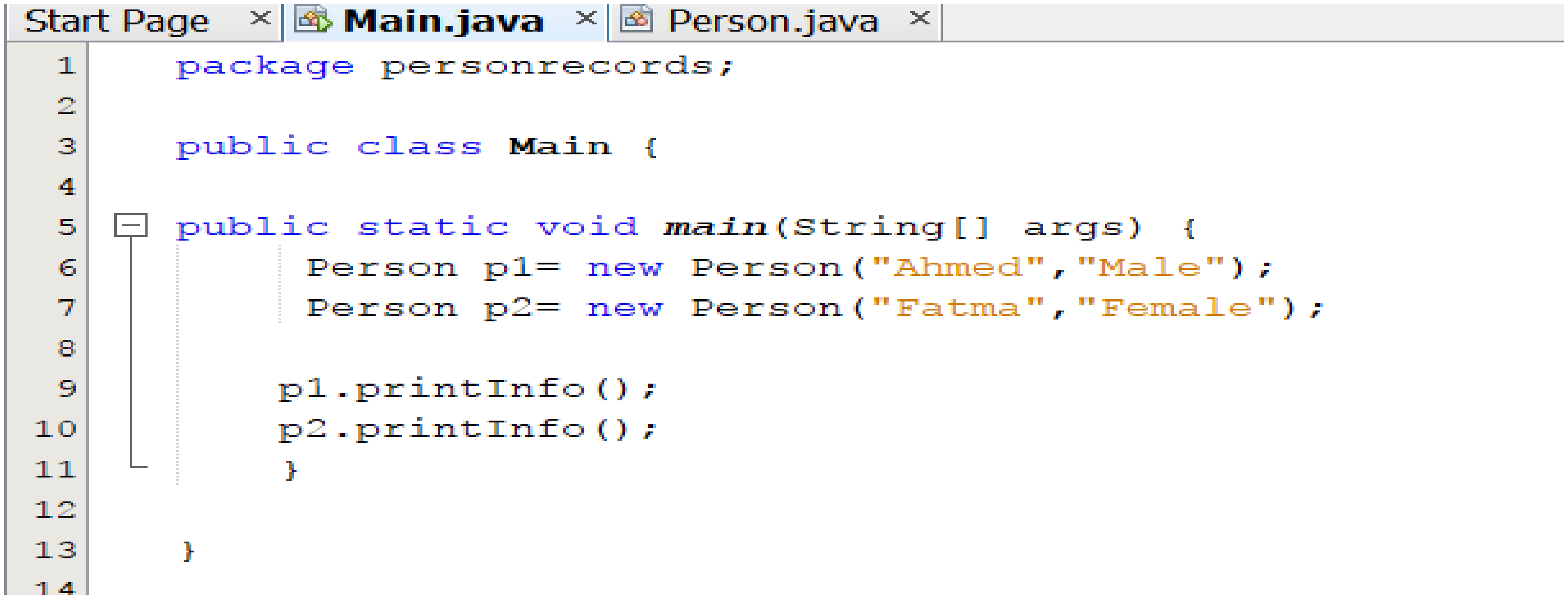
- If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program

this keyword

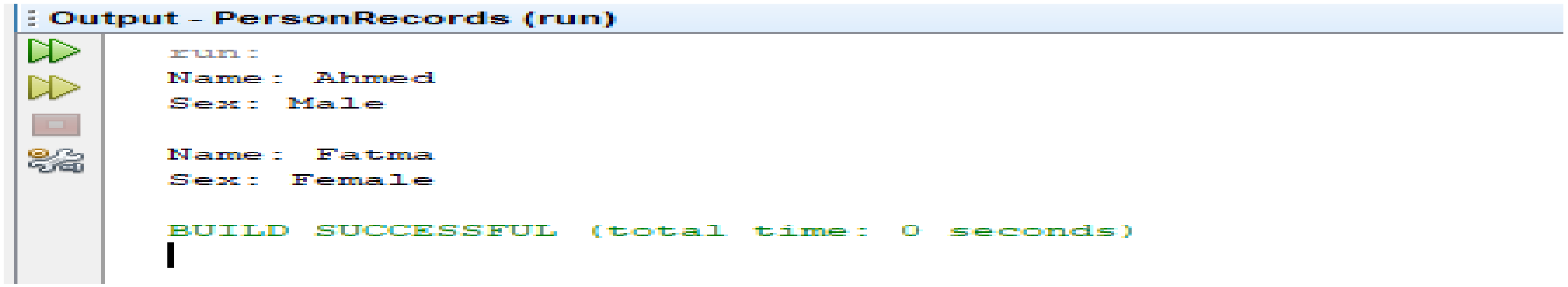


```
package personrecords;

public class Person {

    String name;
     String sex;

     public Person()  {

     }

     public Person(String n,  String s)  {
         name  = n;
         sex   = s;
     }


     void printInfo()  {
         System.out.println("Name:  "  +name);
         System.out.println("Sex:  "   +sex);
         System.out.println();
     }
}
```

this keyword


Start Page × Main.java × Person.java ×

```
package personrecords;

public class Main {

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

      p1.printInfo();
      p2.printInfo();
      }

}
```


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”