Structured Programming

Lecture6

Dr. Obead Alhadreti

Outline

- Protected Members
- Method Overriding
- Method Overloading

Protected Members

Protected Members

- With inheritance, the common instance variables and methods of all the classes in the hierarchy are declared in a superclass.
- Methods of a subclass cannot directly access private members of their superclass. They can be accessed only through the public or protected methods inherited from the superclass.
- Subclass methods can refer to public and protected members inherited from the superclass simply by using the member names.

Protected Members

- Use the protected access modifier when a superclass should provide a method or variable only to its subclasses and other classes in the same package, but not to other classes.

- A superclass's protected members have an intermediate level of protection between public and private access. They can be accessed by members of the superclass, by members of its subclasses and by members of other classes in the same package.

Method Overriding

Method Overriding

- Often a subclass does not want to have the same exact method implementations as it’s superclass. In this case, the subclass can override the methods that it wants to change.

- To override a method, you must re-declare the method in the subclass.

- The new method must have the exact same signature (name, plus the number and the type of its parameters) and return type as the superclass’ method you are overriding.

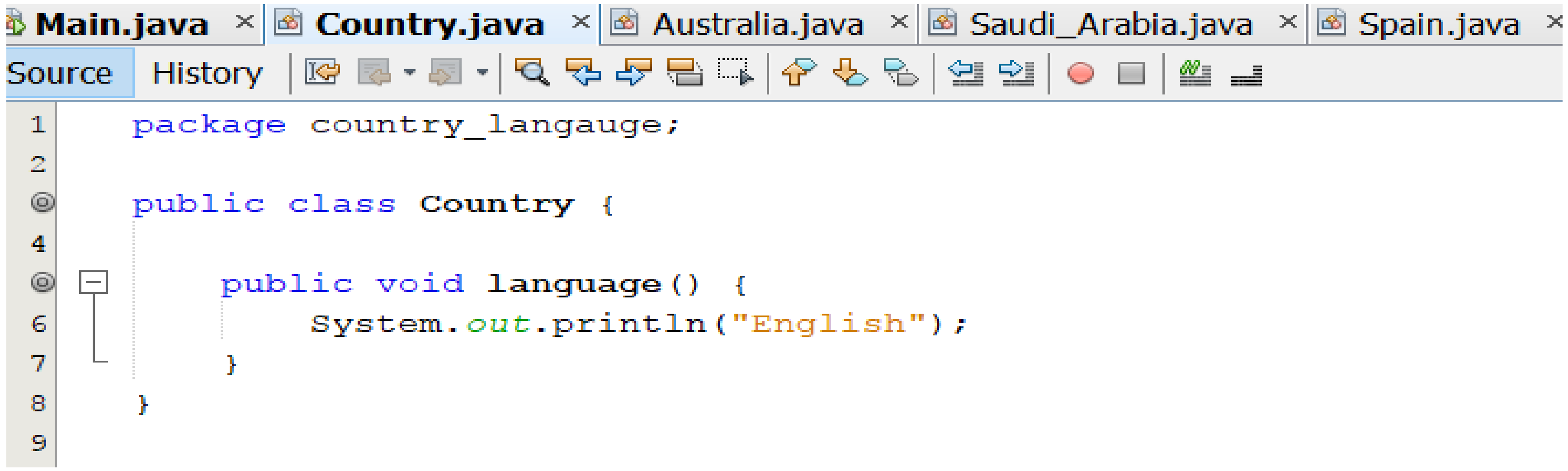
Method Overriding

- Only non-private methods can be overridden because private methods are not visible in the subclass.
- You cannot override static or final methods.
- You cannot override constructors.
- You should put @Override before the overridden method.

Method Overriding

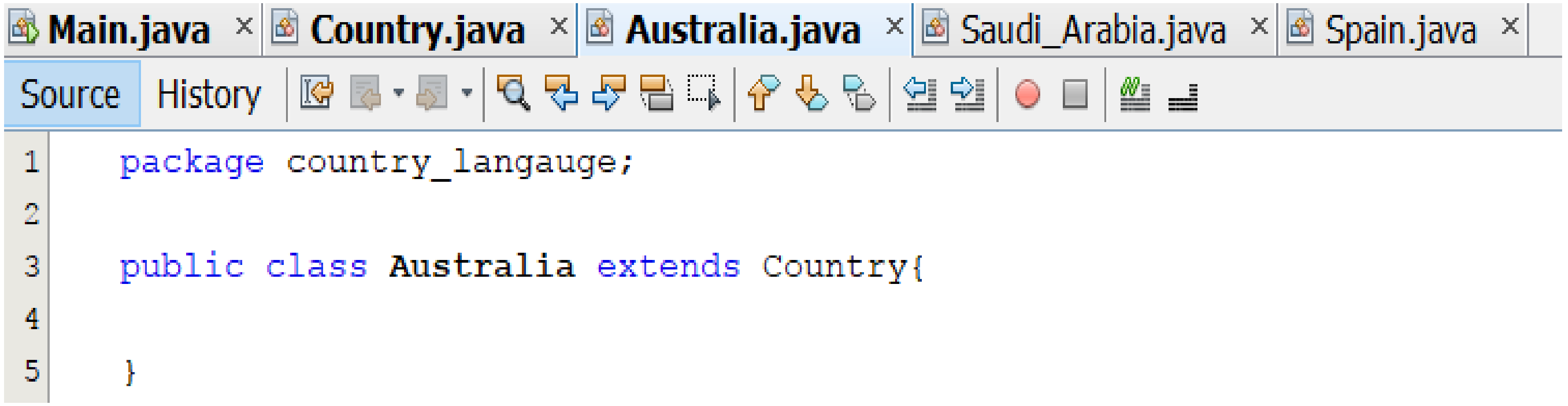
- Methods in the child class must have same name as in the parent class
- Methods in the child class must have same access modifier as in the parent class.
- Methods in the child class must have same parameter as in the parent class.
- Methods in the child class must have same return type as in the parent class.

Example


Main.java | Country.java | Australia.java | Saudi_Arabia.java | Spain.java

Source | History

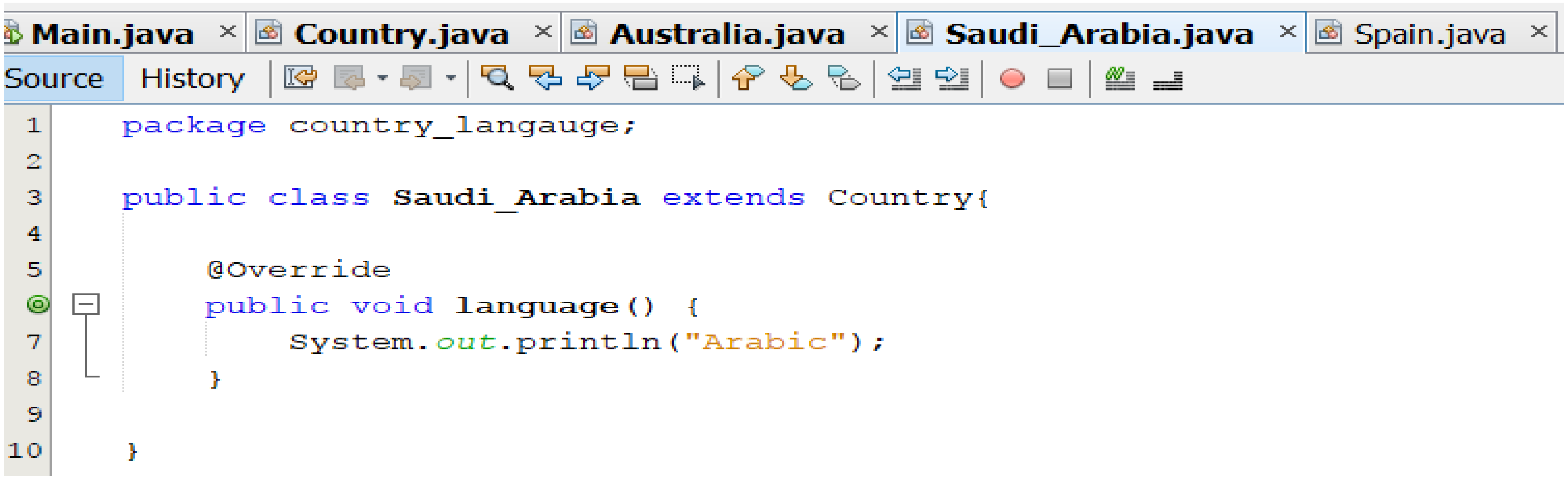
```
package country_langauge;

public class Country {

    public void language()  {
        System.out.println("English");
    }
}

```


Main.java | Country.java | Australia.java | Saudi_Arabia.java | Spain.java

Source | History

```
package country_langauge;

public class Australia extends Country{


}
```

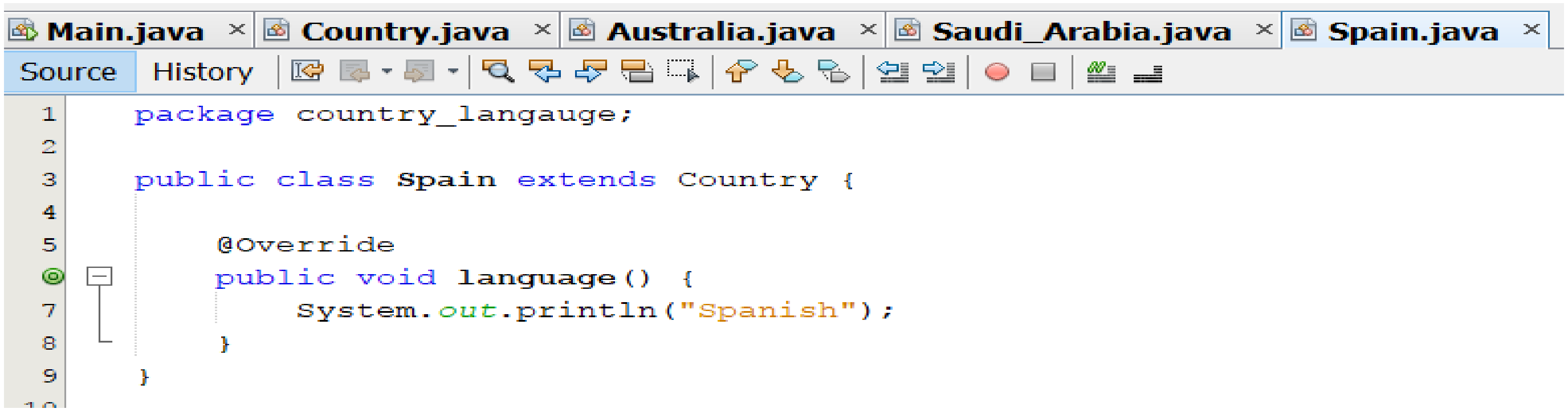
Example



Main.java | Country.java | Australia.java | Saudi_Arabia.java | Spain.java

Source | History

```java
package country_langauge;

public class Saudi_Arabia extends Country{

    @Override
    public void language()  {
        System.out.println("Arabic");
    }

}
```

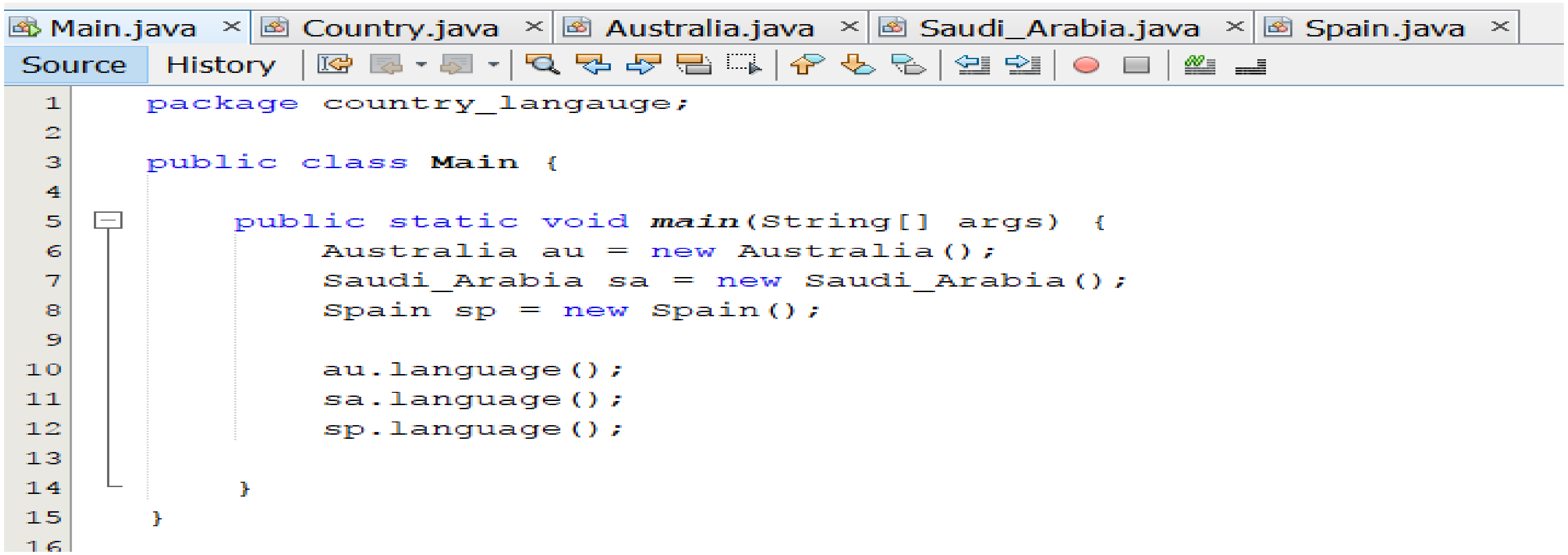


Main.java | Country.java | Australia.java | Saudi_Arabia.java | Spain.java

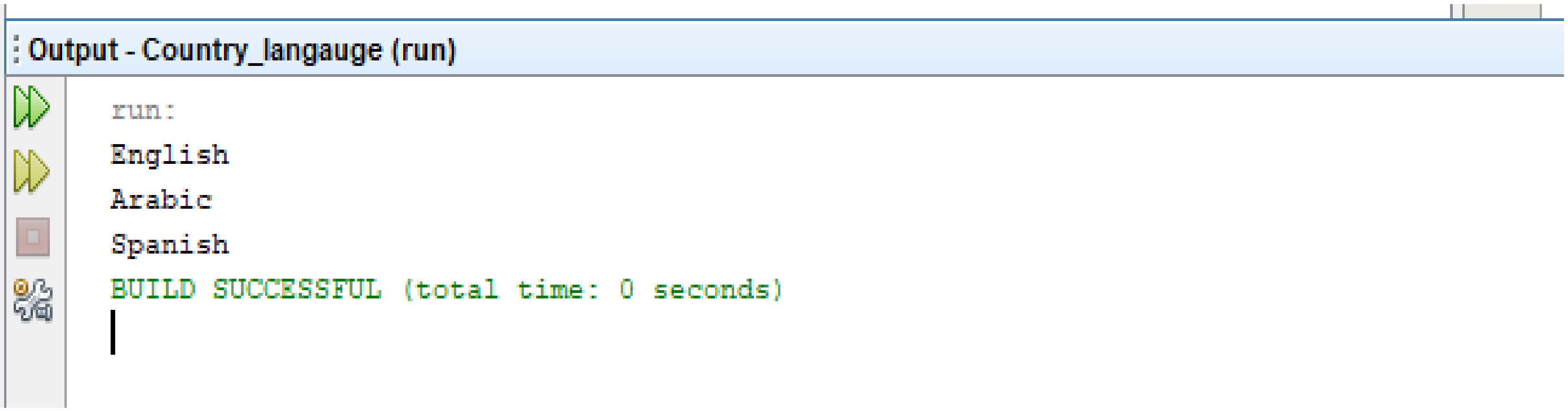
Source | History

```java
package country_langauge;

public class Spain extends Country {

    @Override
    public void language()  {
        System.out.println("Spanish");
    }
}
```

Example



```java
package country_langauge;

public class Main {

    public static void main(String[] args) {
        Australia au = new Australia();
        Saudi_Arabia sa = new Saudi_Arabia();
        Spain sp = new Spain();

        au.language();
        sa.language();
        sp.language();

    }
}
```



```
run:
English
Arabic
Spanish
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method Overloading

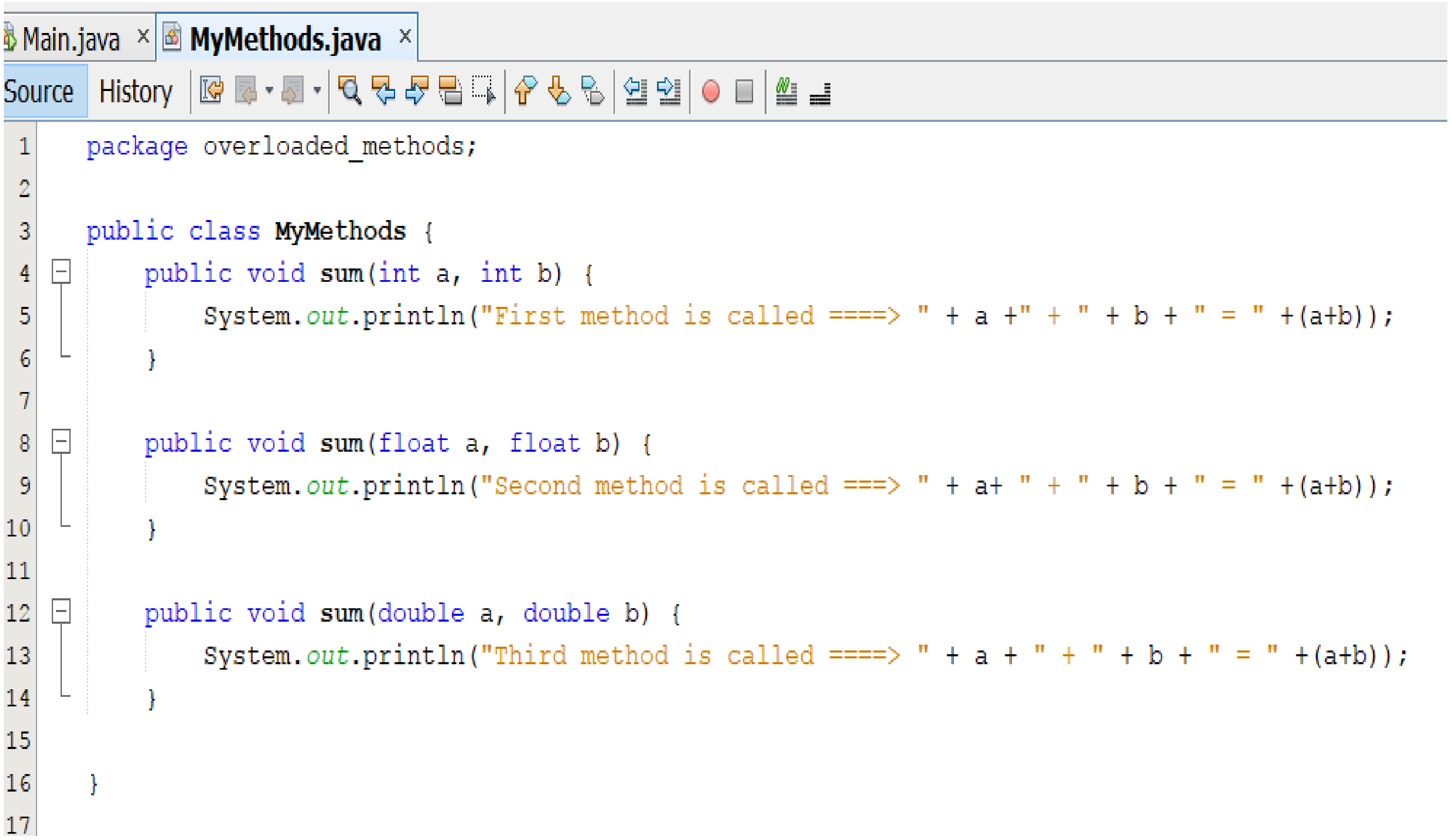
Method Overloading

- Method overloading occur when **methods** or **constructors** of the same name declared in the same class
- Methods overloading is used to create several methods with the same name that perform the same or similar tasks, but on different types or different numbers of parameters.
- Overloaded methods must have different sets of parameters
- Compiler selects the appropriate method to call by examining the number, types and order of the parameters in the call.

Method Overloading

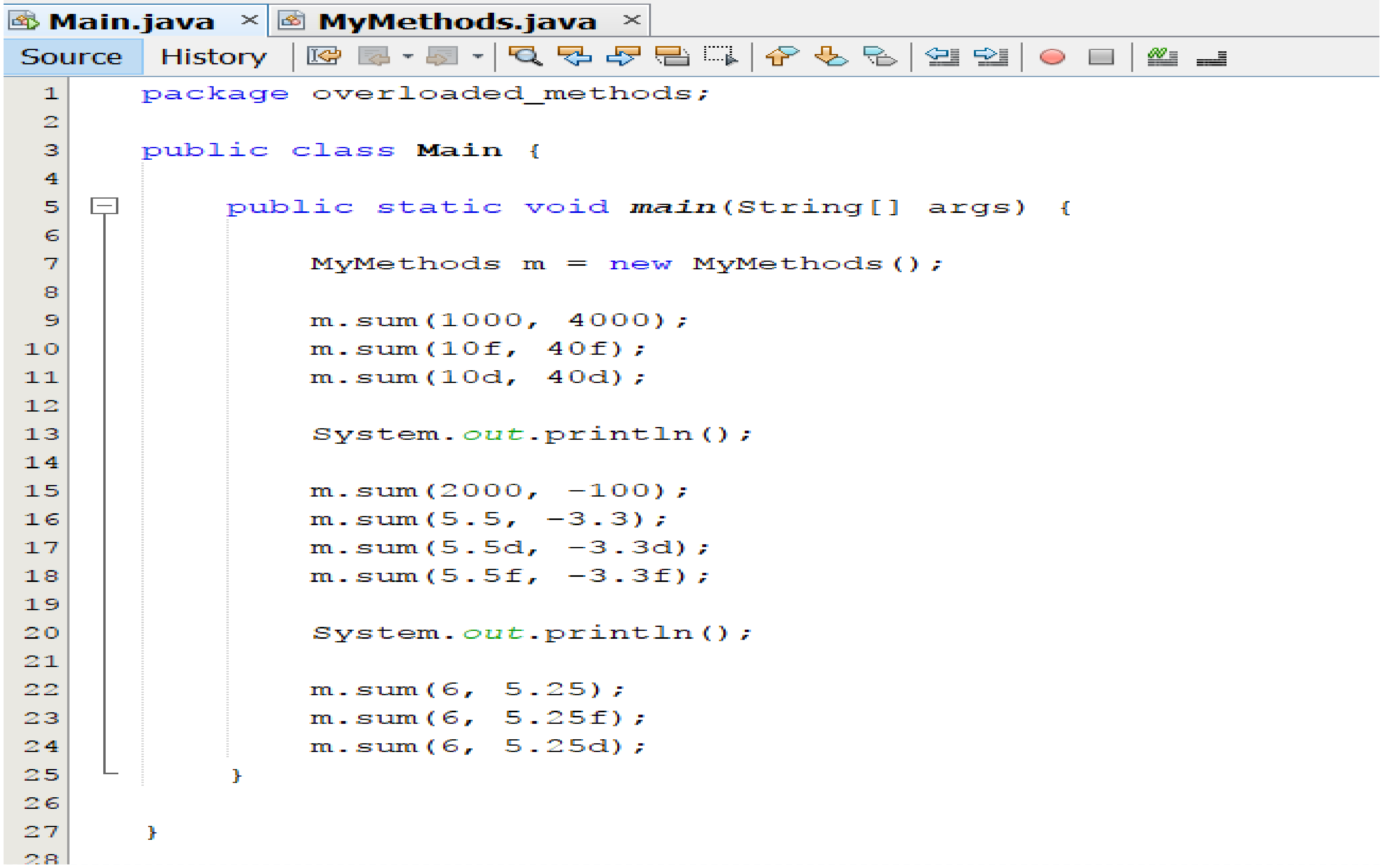
- Overloaded methods can have different return types if the methods have different parameter lists.
- Method calls cannot be distinguished by return type.
- Overloaded methods need not have the same number of parameters.

Ex.1 (Methods which have the same name, but differ in parameter type)



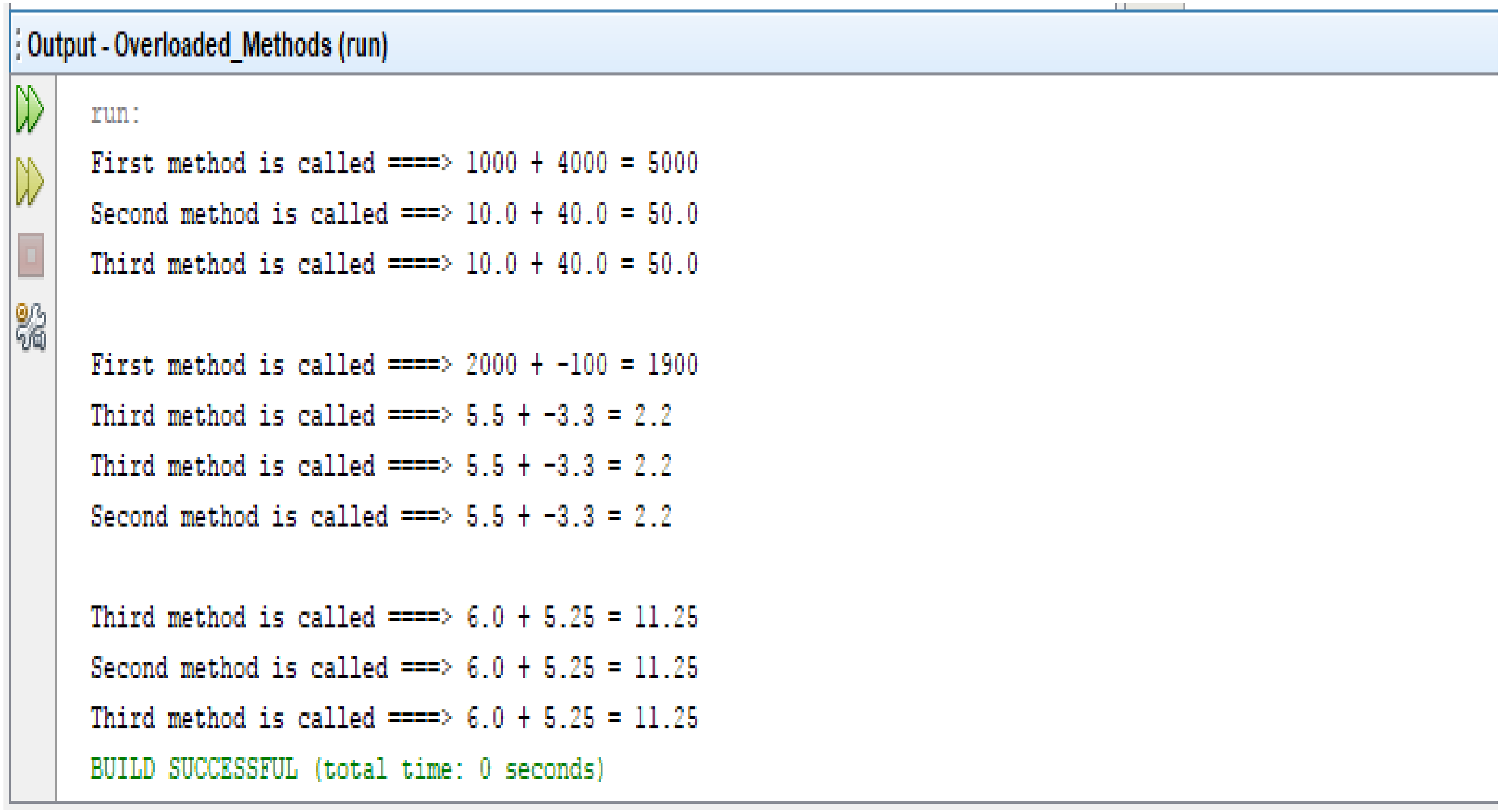
```
package overloaded_methods;

public class MyMethods {
    public void sum(int a, int b) {
        System.out.println("First method is called ====> " + a +" + " + b + " = " +(a+b));
    }

    public void sum(float a, float b) {
        System.out.println("Second method is called ===> " + a+ " + " + b + " = " +(a+b));
    }

    public void sum(double a, double b) {
        System.out.println("Third method is called ====> " + a + " + " + b + " = " +(a+b));
    }

}
```

Ex.1 (Methods which have the same name, but differ in parameter type)



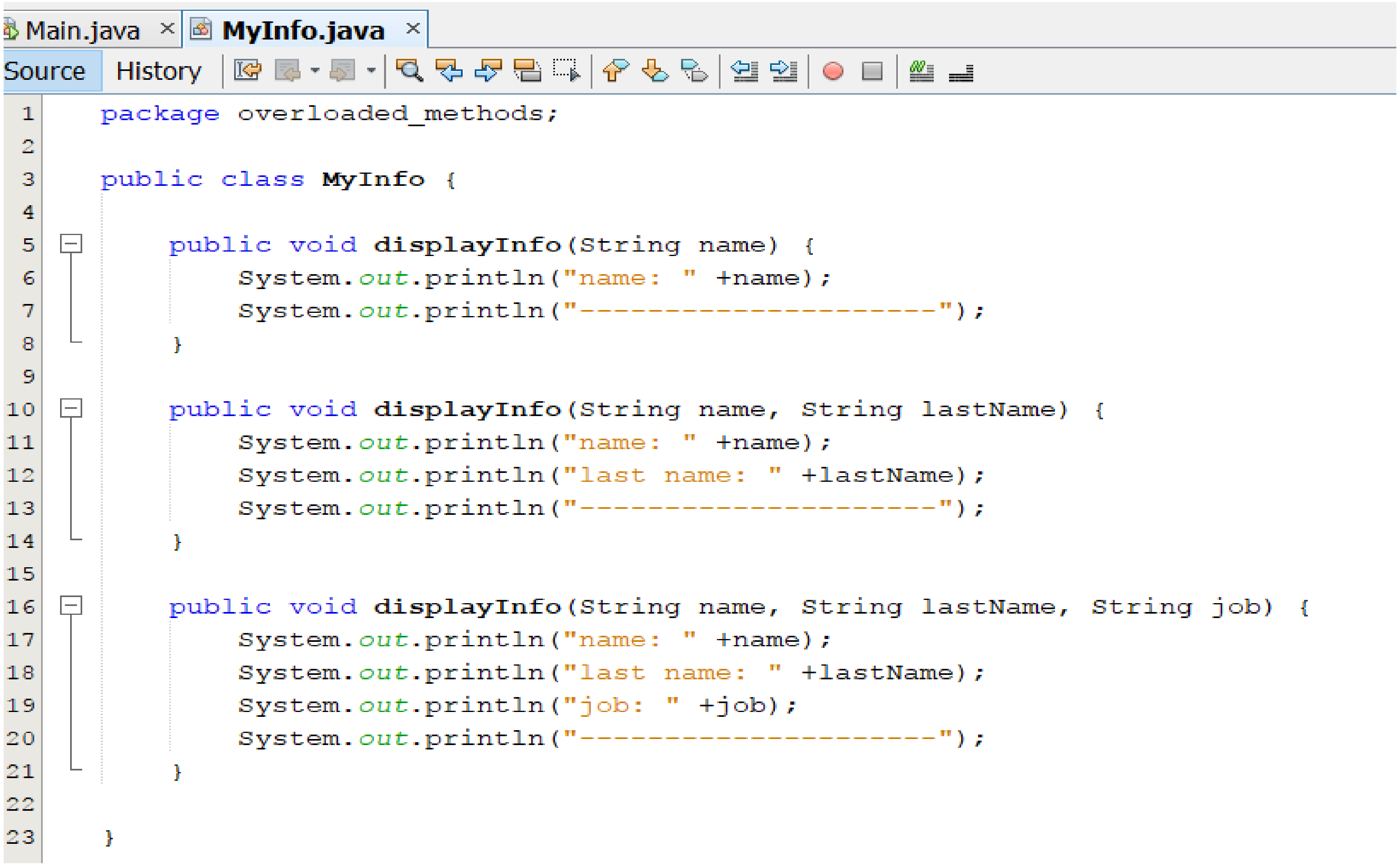
```java
package overloaded_methods;

public class Main {

    public static void main(String[] args) {

        MyMethods m = new MyMethods();

        m.sum(1000, 4000);
        m.sum(10f, 40f);
        m.sum(10d, 40d);

        System.out.println();

        m.sum(2000, -100);
        m.sum(5.5, -3.3);
        m.sum(5.5d, -3.3d);
        m.sum(5.5f, -3.3f);

        System.out.println();

        m.sum(6, 5.25);
        m.sum(6, 5.25f);
        m.sum(6, 5.25d);
    }

}
```

Ex.1 (Methods which have the same name, but differ in parameter type)



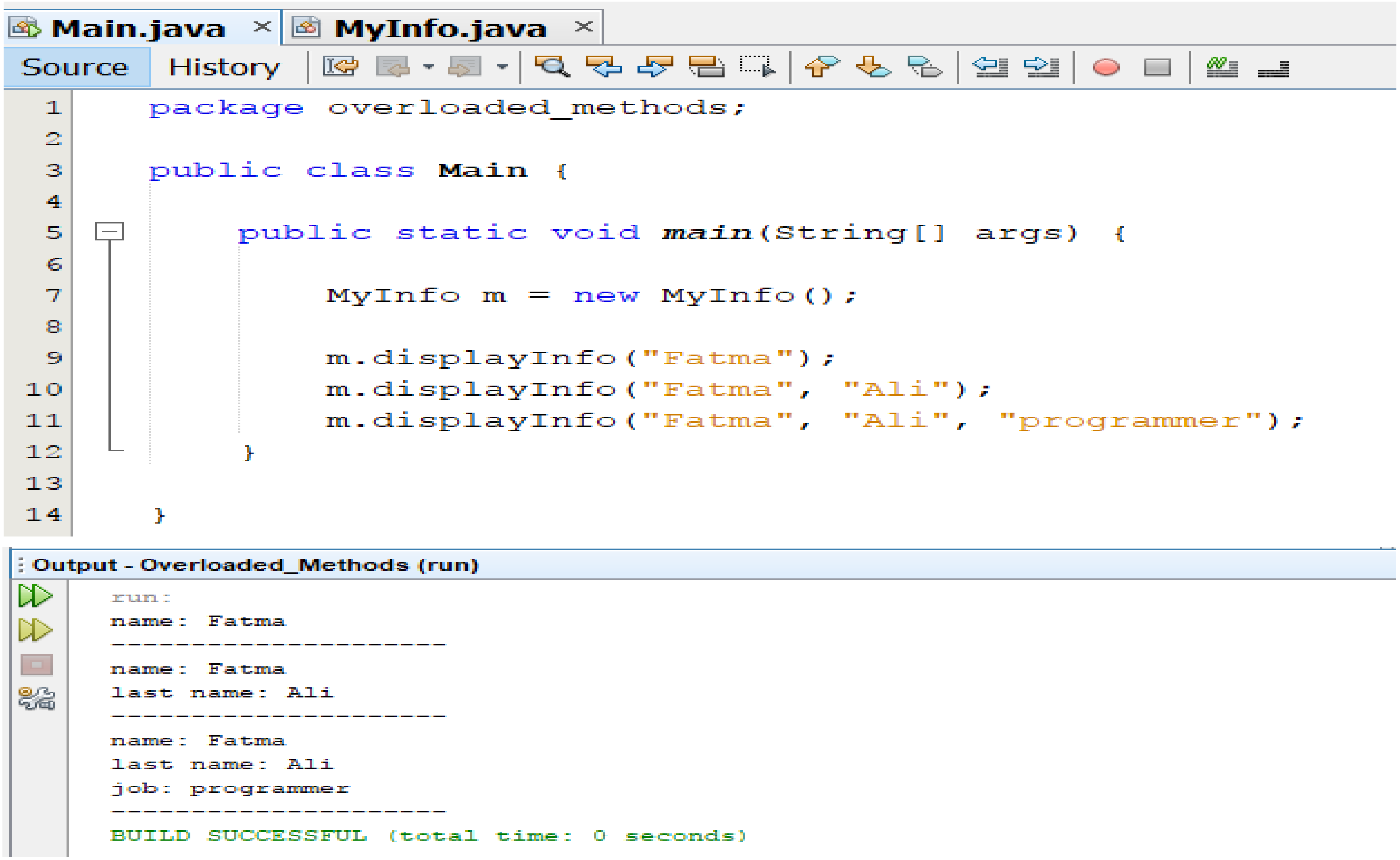
```
run:
First method is called ====> 1000 + 4000 = 5000
Second method is called ===> 10.0 + 40.0 = 50.0
Third method is called ====> 10.0 + 40.0 = 50.0

First method is called ====> 2000 + -100 = 1900
Third method is called ====> 5.5 + -3.3 = 2.2
Third method is called ====> 5.5 + -3.3 = 2.2
Second method is called ===> 5.5 + -3.3 = 2.2

Third method is called ====> 6.0 + 5.25 = 11.25
Second method is called ===> 6.0 + 5.25 = 11.25
Third method is called ====> 6.0 + 5.25 = 11.25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ex.2 (Methods which have the same name, but differ in parameter number)



```
package overloaded_methods;

public class MyInfo {

    public void displayInfo(String name) {
        System.out.println("name:  " +name);
        System.out.println("---------------------");
    }

    public void displayInfo(String name, String lastName) {
        System.out.println("name:  " +name);
        System.out.println("last name:  " +lastName);
        System.out.println("---------------------");
    }

    public void displayInfo(String name, String lastName, String job) {
        System.out.println("name:  " +name);
        System.out.println("last name:  " +lastName);
        System.out.println("job:  " +job);
        System.out.println("---------------------");
    }

}
```

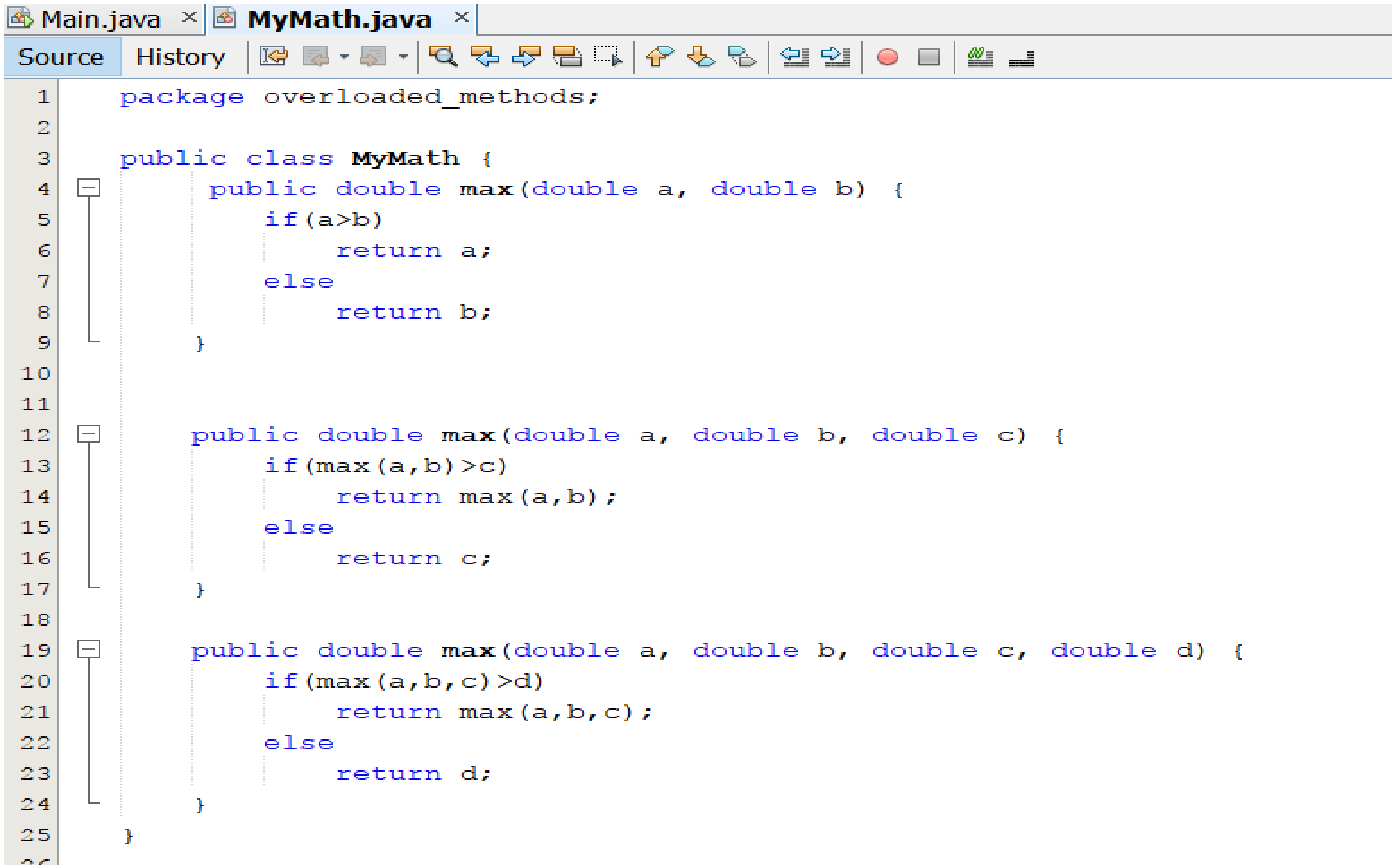
Ex.2 (Methods which have the same name, but differ in parameter number)



```java
package overloaded_methods;

public class Main {

    public static void main(String[] args) {

        MyInfo m = new MyInfo();

        m.displayInfo("Fatma");
        m.displayInfo("Fatma", "Ali");
        m.displayInfo("Fatma", "Ali", "programmer");
    }

}
```

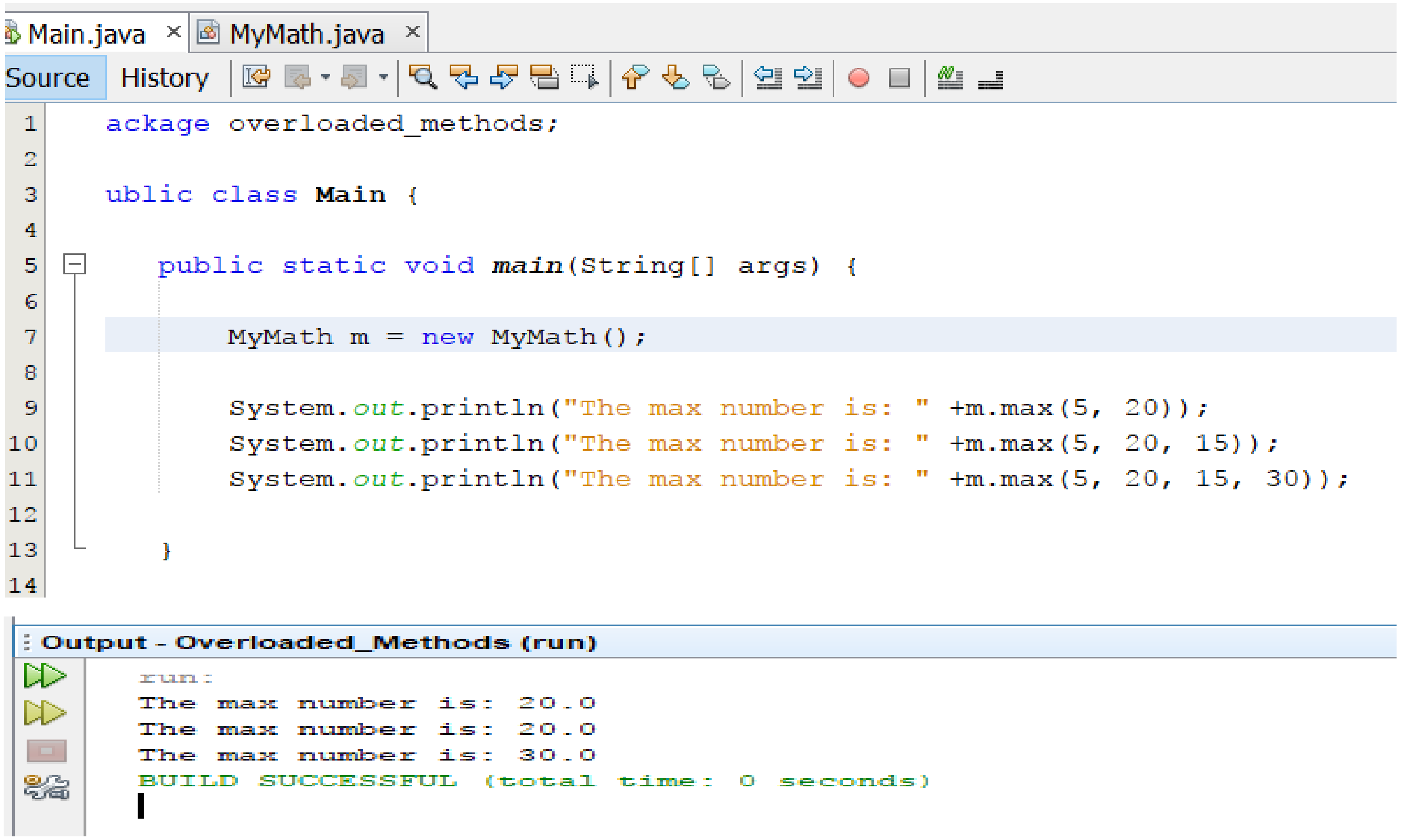
```
run:
name: Fatma
----------------------
name: Fatma
last name: Ali
----------------------
name: Fatma
last name: Ali
job: programmer
----------------------
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ex.3 (Methods which rely on existing methods)



```java
package overloaded_methods;

public class MyMath {
    public double max(double a, double b) {
        if(a>b)
            return a;
        else
            return b;
    }


    public double max(double a, double b, double c) {
        if(max(a,b)>c)
            return max(a,b);
        else
            return c;
    }

    public double max(double a, double b, double c, double d) {
        if(max(a,b,c)>d)
            return max(a,b,c);
        else
            return d;
    }
}
```

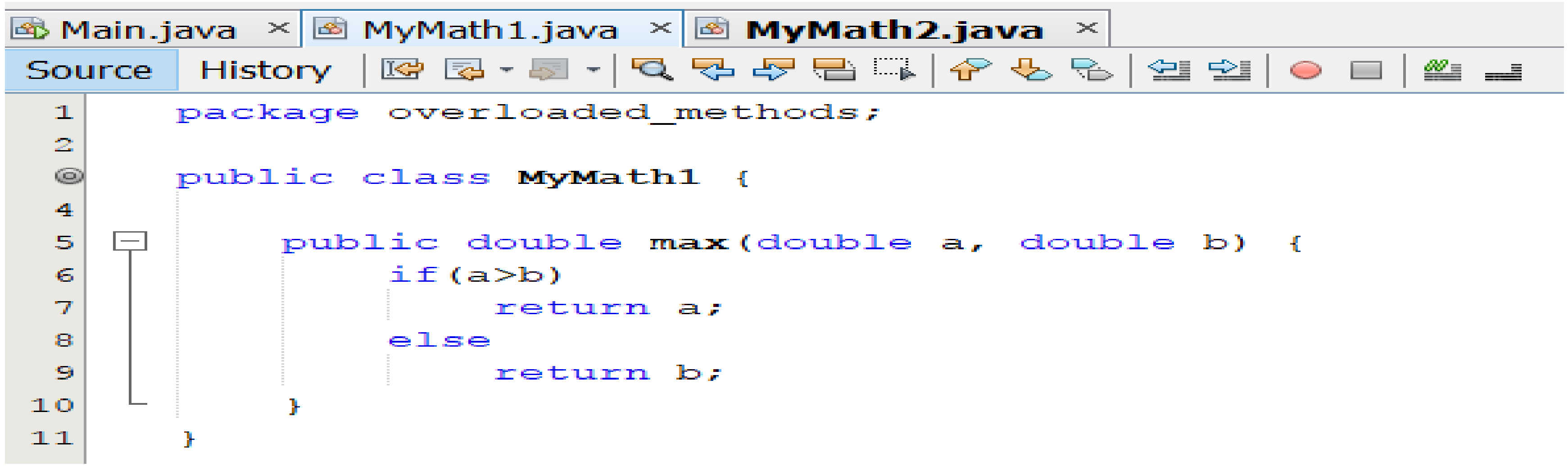
Ex.3 (Methods which rely on existing methods)



```
ackage overloaded_methods;

ublic class Main {

    public static void main(String[] args) {

        MyMath m = new MyMath();

        System.out.println("The max number is: " +m.max(5, 20));
        System.out.println("The max number is: " +m.max(5, 20, 15));
        System.out.println("The max number is: " +m.max(5, 20, 15, 30));

    }

```

```
run:
The max number is: 20.0
The max number is: 20.0
The max number is: 30.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

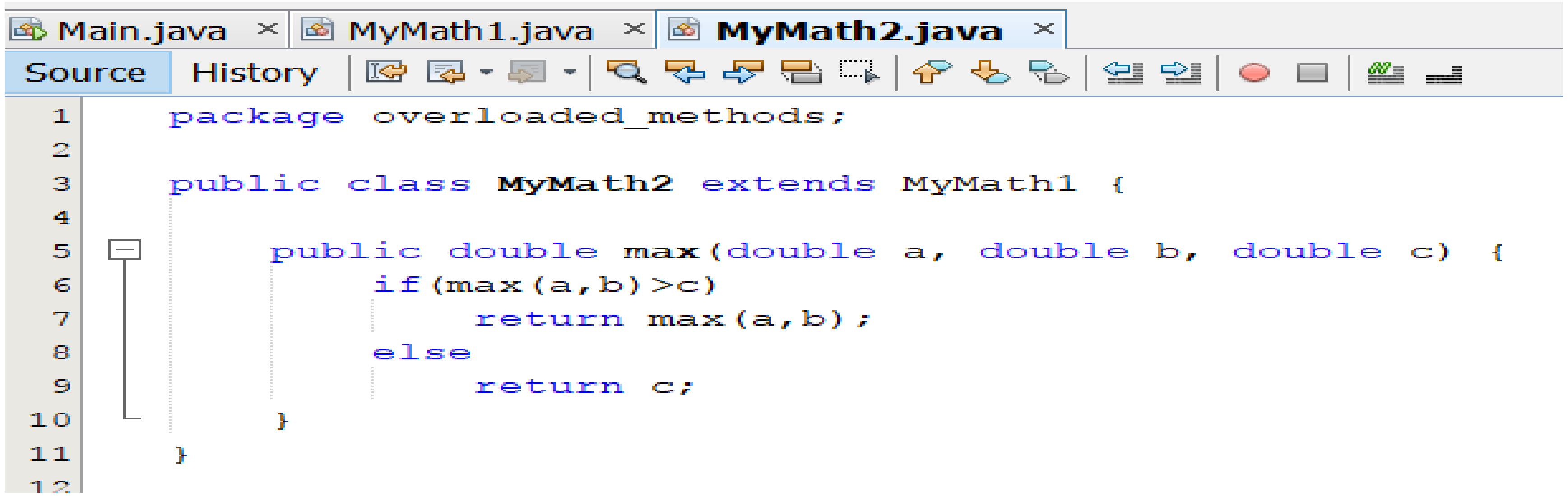
Ex.4 (Methods in subclass which rely on existing methods in superclass)



Main.java × MyMath1.java × MyMath2.java ×

Source History

```
package overloaded_methods;

public class MyMath1 {

    public double max(double a, double b) {
        if(a>b)
            return a;
        else
            return b;
    }
}
```

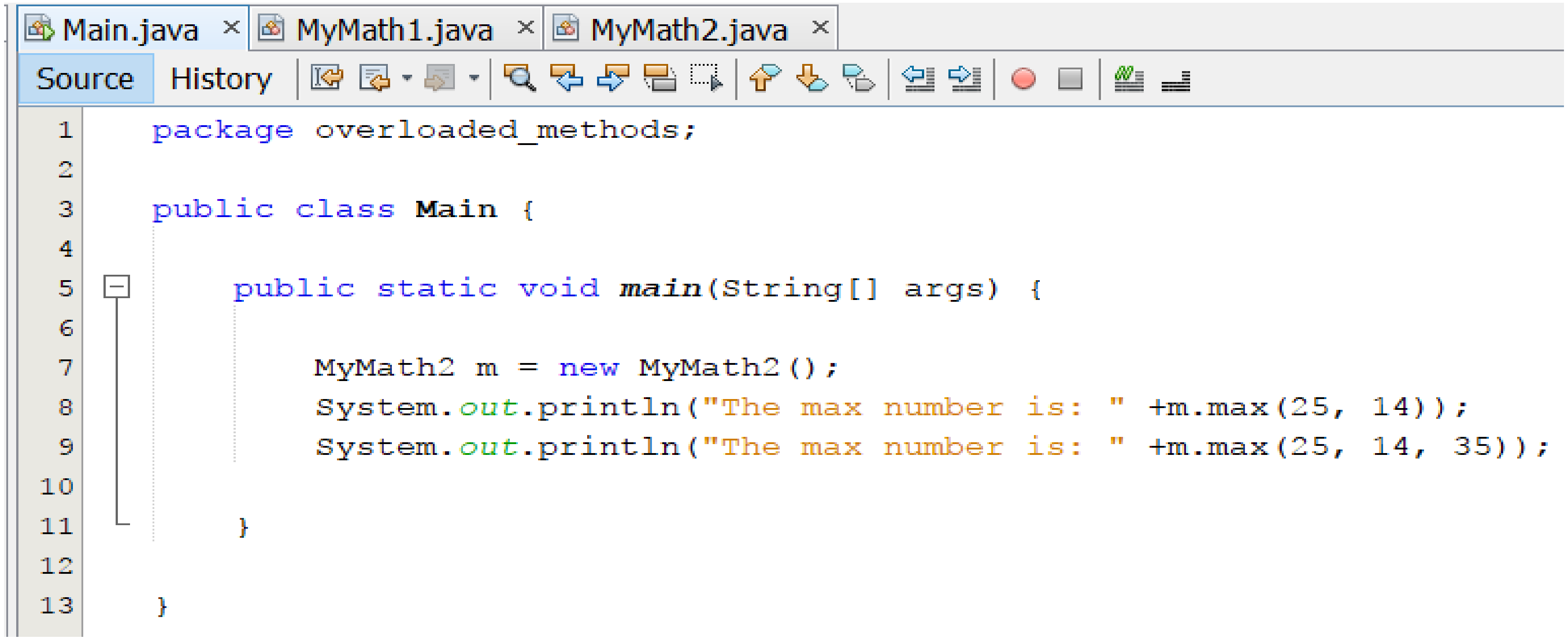


Main.java × MyMath1.java × MyMath2.java ×

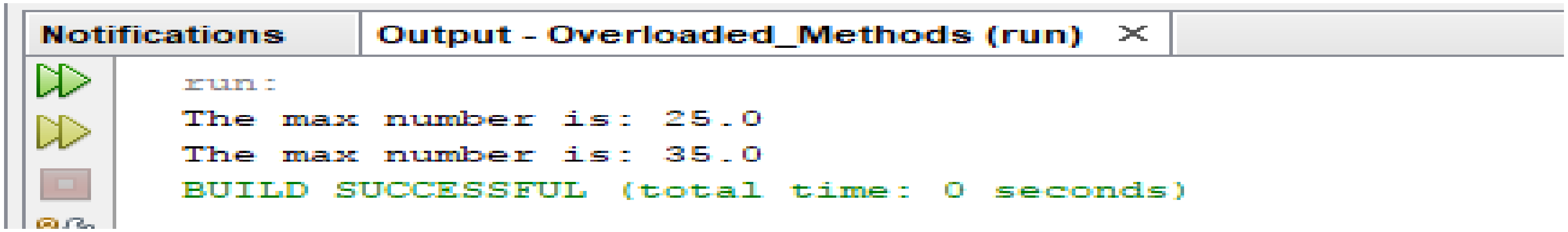
Source History

```
package overloaded_methods;

public class MyMath2 extends MyMath1 {

    public double max(double a, double b, double c) {
        if(max(a,b)>c)
            return max(a,b);
        else
            return c;
    }
}
```

Ex.4 (Methods in subclass which rely on existing methods in superclass)

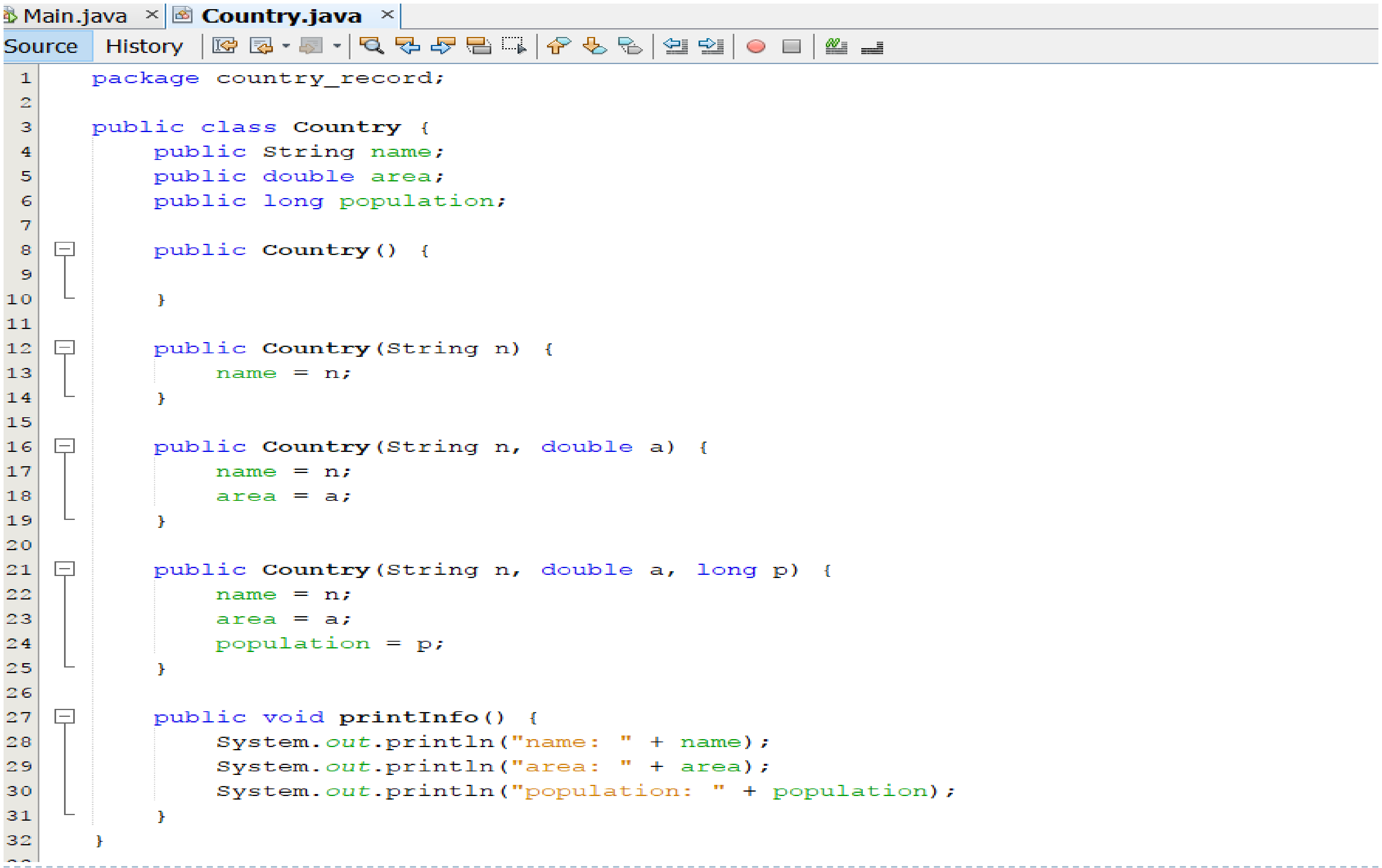


```java
package overloaded_methods;

public class Main {

    public static void main(String[] args) {

        MyMath2 m = new MyMath2();
        System.out.println("The max number is: " +m.max(25, 14));
        System.out.println("The max number is: " +m.max(25, 14, 35));

    }

}
```



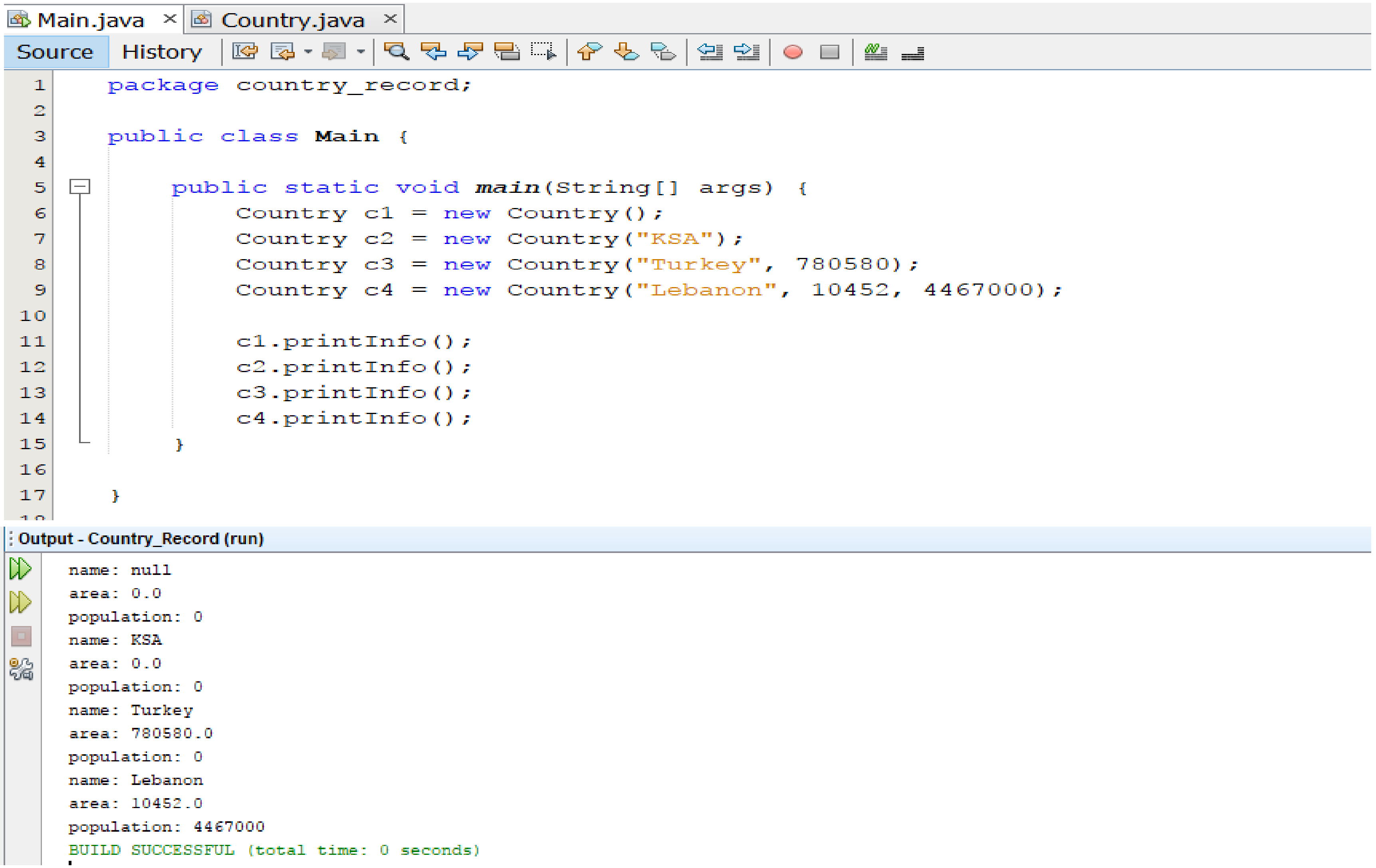
```
run:
The max number is: 25.0
The max number is: 35.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ex.5 (Declaring more than one constructor in the same class)



```java
package country_record;

public class Country {
    public String name;
    public double area;
    public long population;

    public Country() {

    }

    public Country(String n) {
        name = n;
    }

    public Country(String n, double a) {
        name = n;
        area = a;
    }

    public Country(String n, double a, long p) {
        name = n;
        area = a;
        population = p;
    }

    public void printInfo() {
        System.out.println("name: " + name);
        System.out.println("area: " + area);
        System.out.println("population: " + population);
    }
}
```

Ex.5 (Declaring more than one constructor in the same class)



```java
package country_record;

public class Main {

    public static void main(String[] args) {
        Country c1 = new Country();
        Country c2 = new Country("KSA");
        Country c3 = new Country("Turkey", 780580);
        Country c4 = new Country("Lebanon", 10452, 4467000);

        c1.printInfo();
        c2.printInfo();
        c3.printInfo();
        c4.printInfo();
    }

}
```

```
name: null
area: 0.0
population: 0
name: KSA
area: 0.0
population: 0
name: Turkey
area: 780580.0
population: 0
name: Lebanon
area: 10452.0
population: 4467000
BUILD SUCCESSFUL (total time: 0 seconds)
```