

# Topics Covered

- Java variables and methods
- Java control statements
- Java operators

# Java Language Basics

## Variables

- **Local Variables**
- A method will often store its **temporary state** in *local variables*.
- The syntax for declaring a local variable is simple (for example, **int count = 0;**).
- Variable **visibility is where variable is declared** — which is between the opening and closing braces of a method.
- Local variables are **only visible to the methods in which they are declared**; they are not accessible from the rest of the class.

# Java Language Basics

## Variables

- **Local Variables**
- The syntax for declaring a local variable is simple (for example, `int count = 0;`).
- Variables has **two parts**:
- **The data type of the variable**: which types of data values can be stored.
- **Name of the variable**: name used for the variable in the program

# Java Language Basics

## Naming of Variables

- Variable names are case-sensitive.
- A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "\$", or the underscore character "\_". White space is not permitted.

# Java Language Basics

## Data Types of Variables

- A variable's data type determines the values it may contain, plus the operations that may be performed on it.
- A primitive type is predefined by the language and is named by a reserved keyword.
- Java programming language supports eight *primitive data types*.

# Java Language Basics

## Data Types of Variables

Eight *primitive data types*

- For storing Integer: Byte, Short, Int, Long
- For storing real: Float, Double
- For storing characters: char
- For true/false: Boolean

# Java Language Basics

## Data Types of Variables

### int

- The int data type is a 32-bit signed two's complement integer. It has a **minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive)**.
- **For integral values, this data type is generally the default choice.**

# Java Language Basics

## Data Types of Variables

### double

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- For decimal values, this data type is generally the default choice.



# Java Language Basics

## Data Types of Variables

### boolean

- The boolean data type has only two possible values: **true and false**.
- Use this data type for **simple flags that track true/false conditions**.
- This data type represents one bit of information, but its "size" isn't something that's precisely defined.

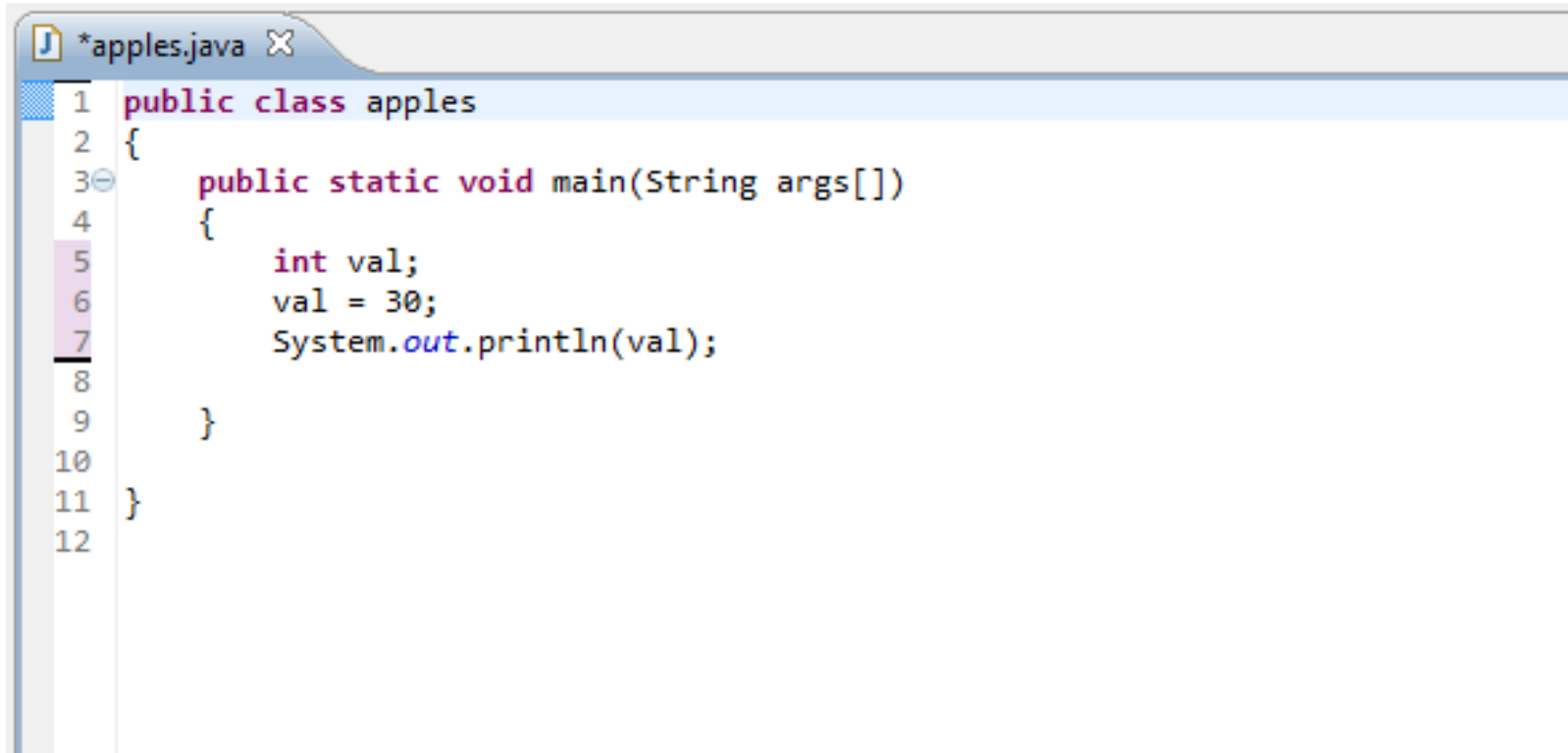
# Java Language Basics

## Data Types of Variables

### char

- The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).
- Used for storing characters, e.g., 'A', 'E', '2', etc.

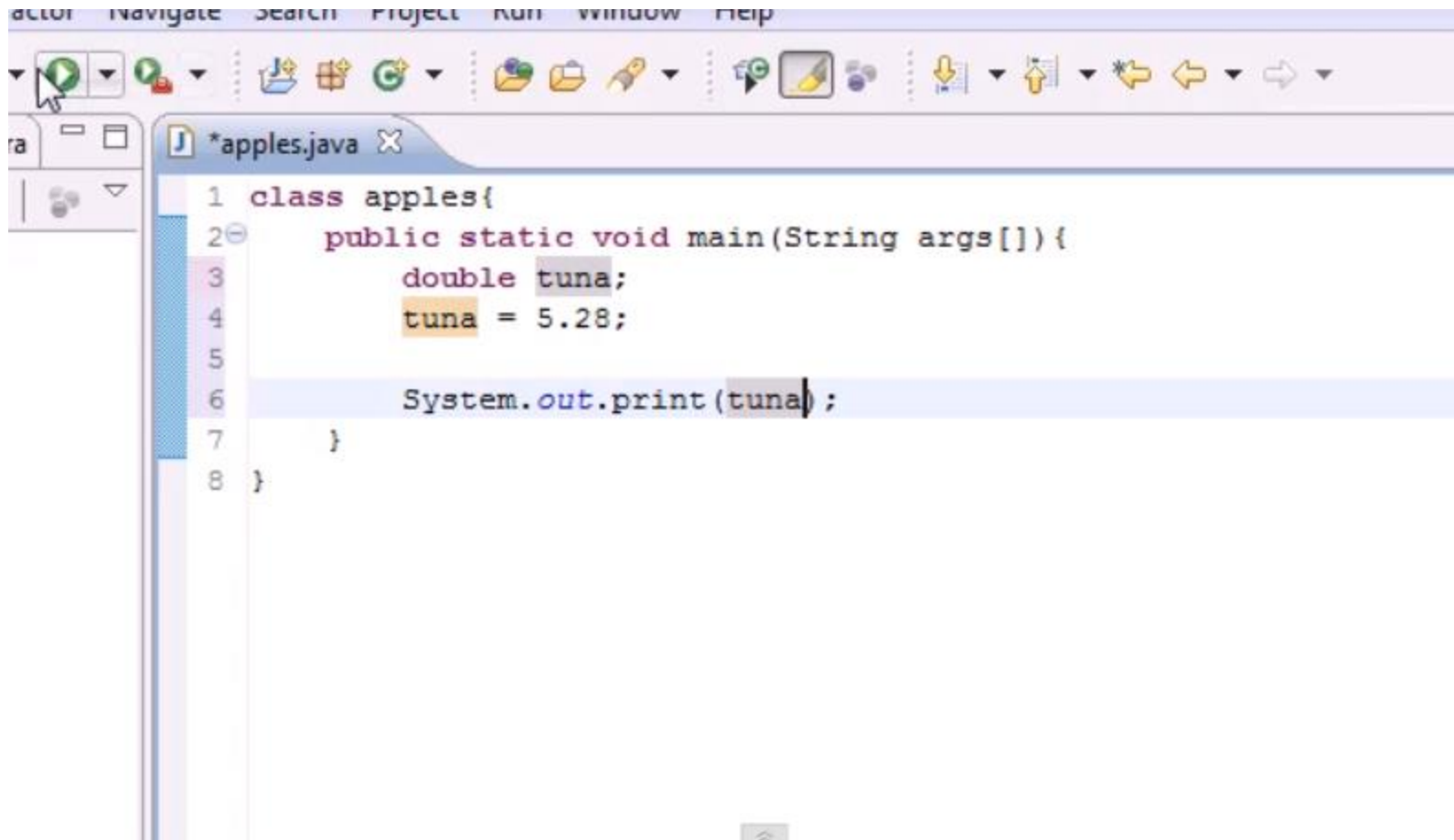
# Introduction to variables in Java



```
1 public class apples
2 {
3     public static void main(String args[])
4     {
5         int val;
6         val = 30;
7         System.out.println(val);
8     }
9 }
10
11
12
```

The screenshot shows a Java IDE window titled '\*apples.java'. The code is a simple Java class named 'apples' with a 'main' method. Inside the 'main' method, an integer variable 'val' is declared, assigned the value 30, and then printed to the console using 'System.out.println(val);'. The code is formatted with standard Java syntax highlighting: keywords in purple, identifiers in black, and literals in blue.

# Introduction to variables in Java

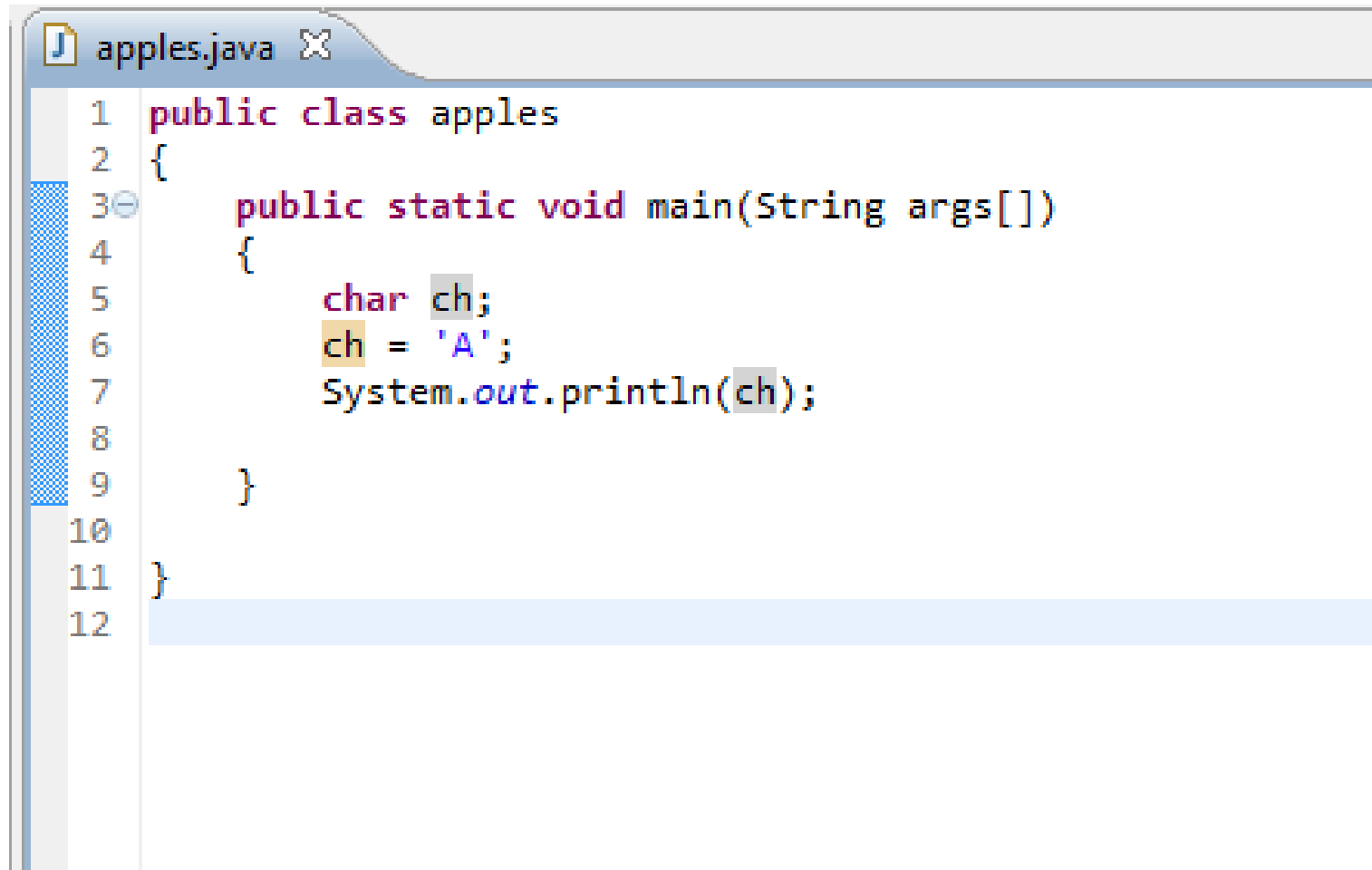


The screenshot shows an IDE window titled '\*apples.java'. The code is as follows:

```
1 class apples{  
2     public static void main(String args[]) {  
3         double tuna;  
4         tuna = 5.28;  
5  
6         System.out.print(tuna);  
7     }  
8 }
```

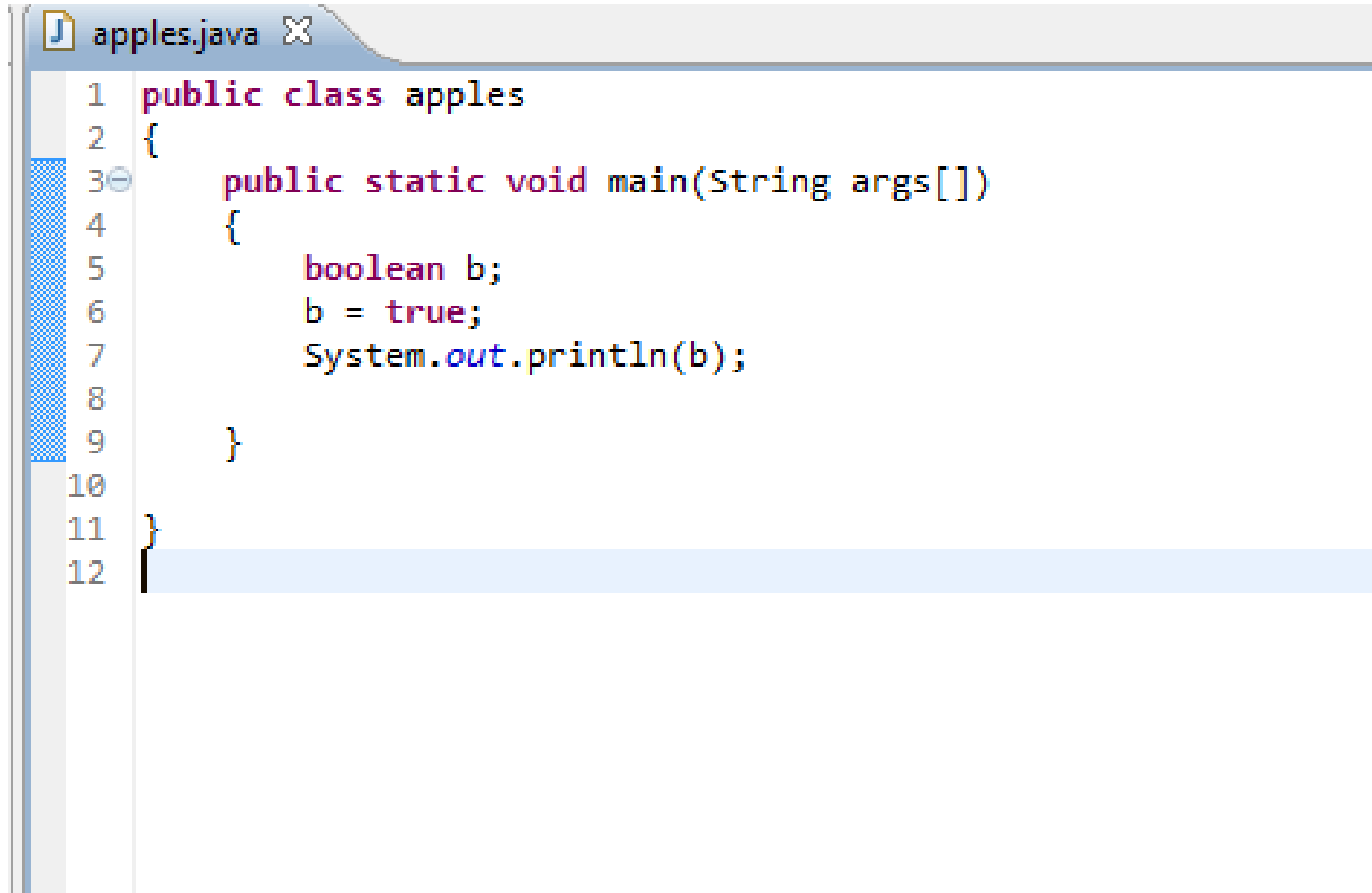
The variable `tuna` is declared as a `double` on line 3, assigned the value `5.28` on line 4, and printed on line 6. The IDE interface includes a menu bar (Editor, Navigate, Search, Project, Run, Window, Help) and a toolbar with various icons.

# Introduction to variables in Java



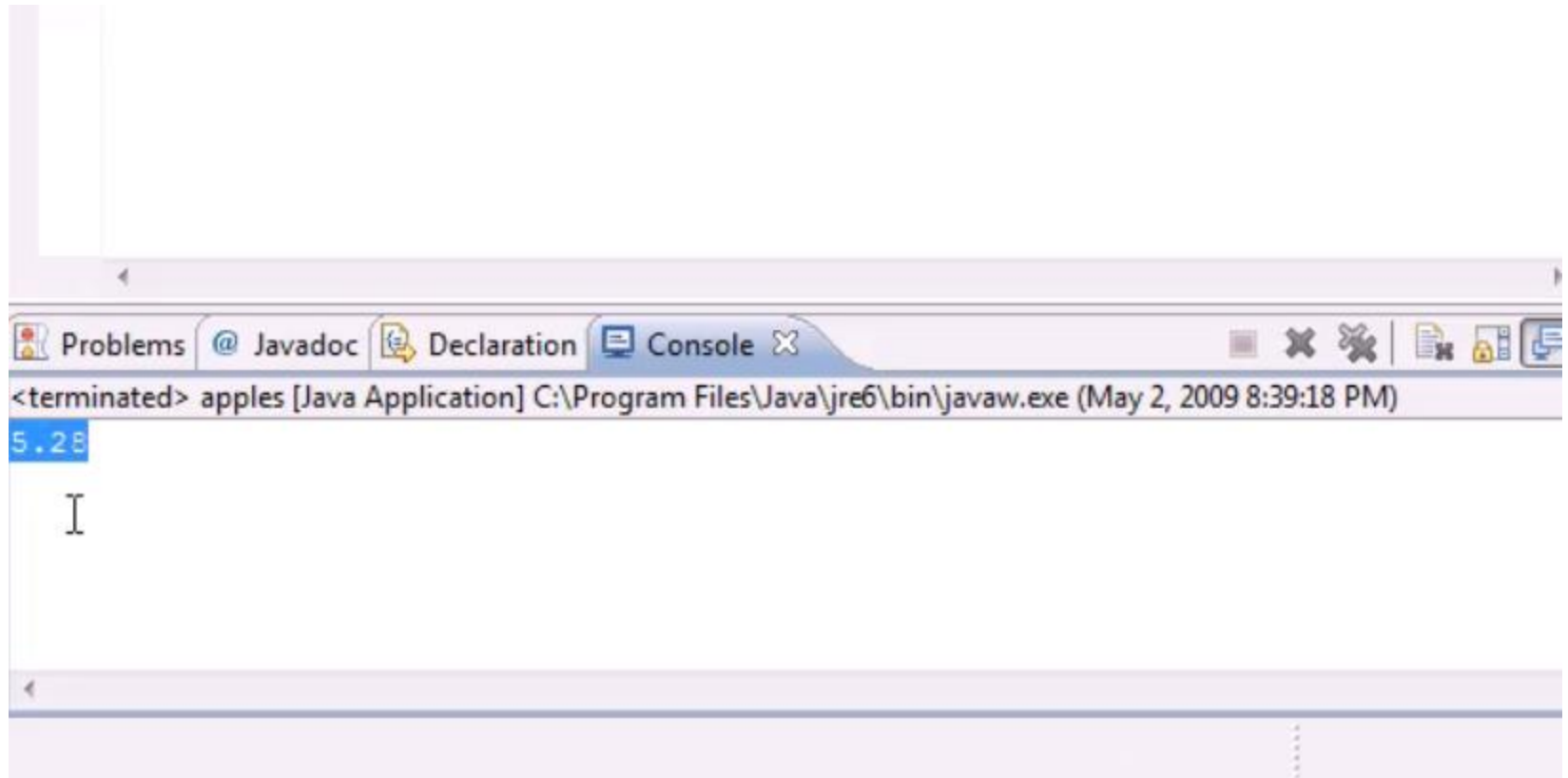
```
apples.java X
1 public class apples
2 {
3     public static void main(String args[])
4     {
5         char ch;
6         ch = 'A';
7         System.out.println(ch);
8     }
9 }
10
11
12
```

# Introduction to variables in Java

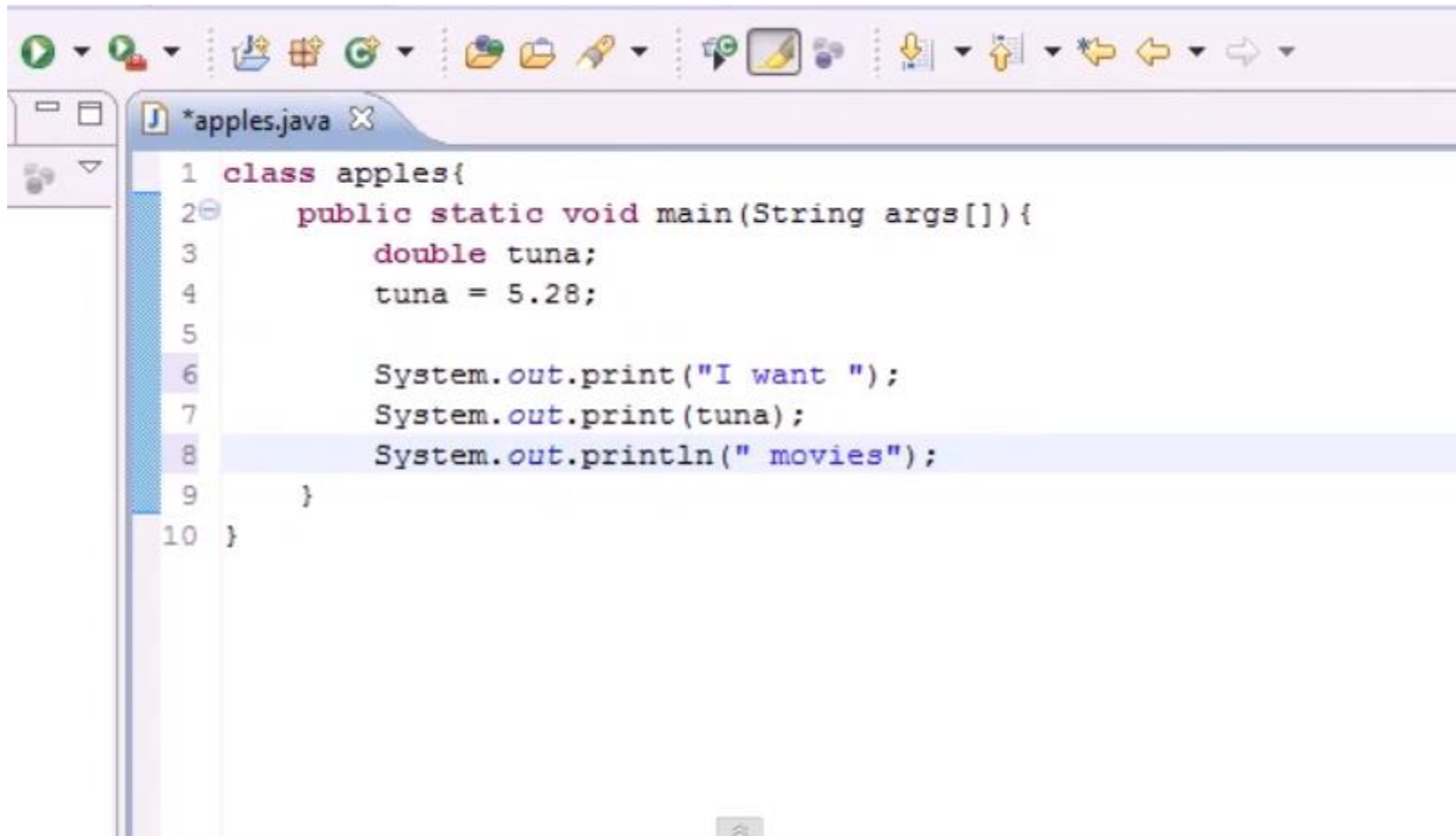


```
apples.java X
1 public class apples
2 {
3     public static void main(String args[])
4     {
5         boolean b;
6         b = true;
7         System.out.println(b);
8     }
9 }
10
11
12
```

# View output



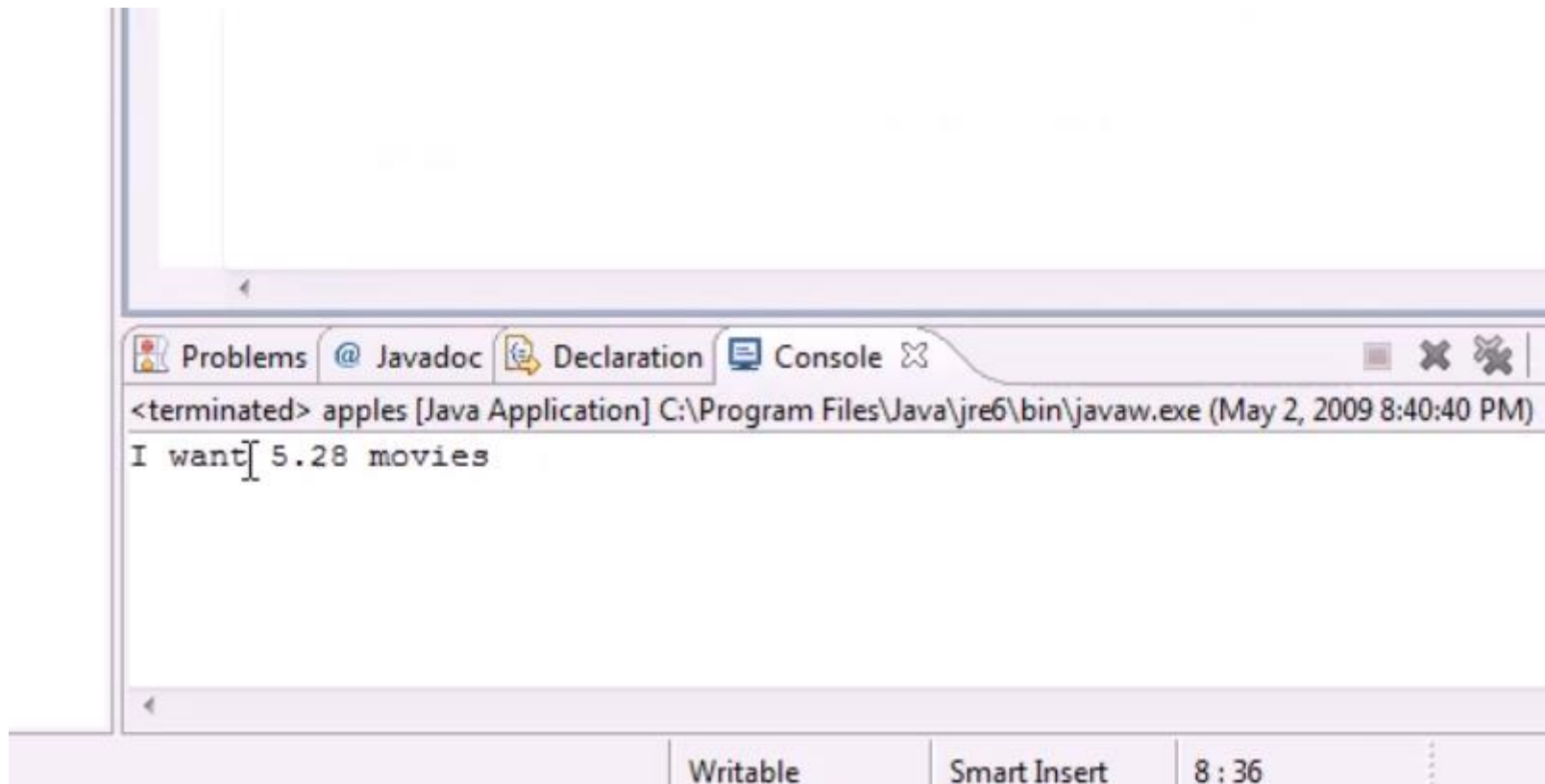
# Print and println in Java



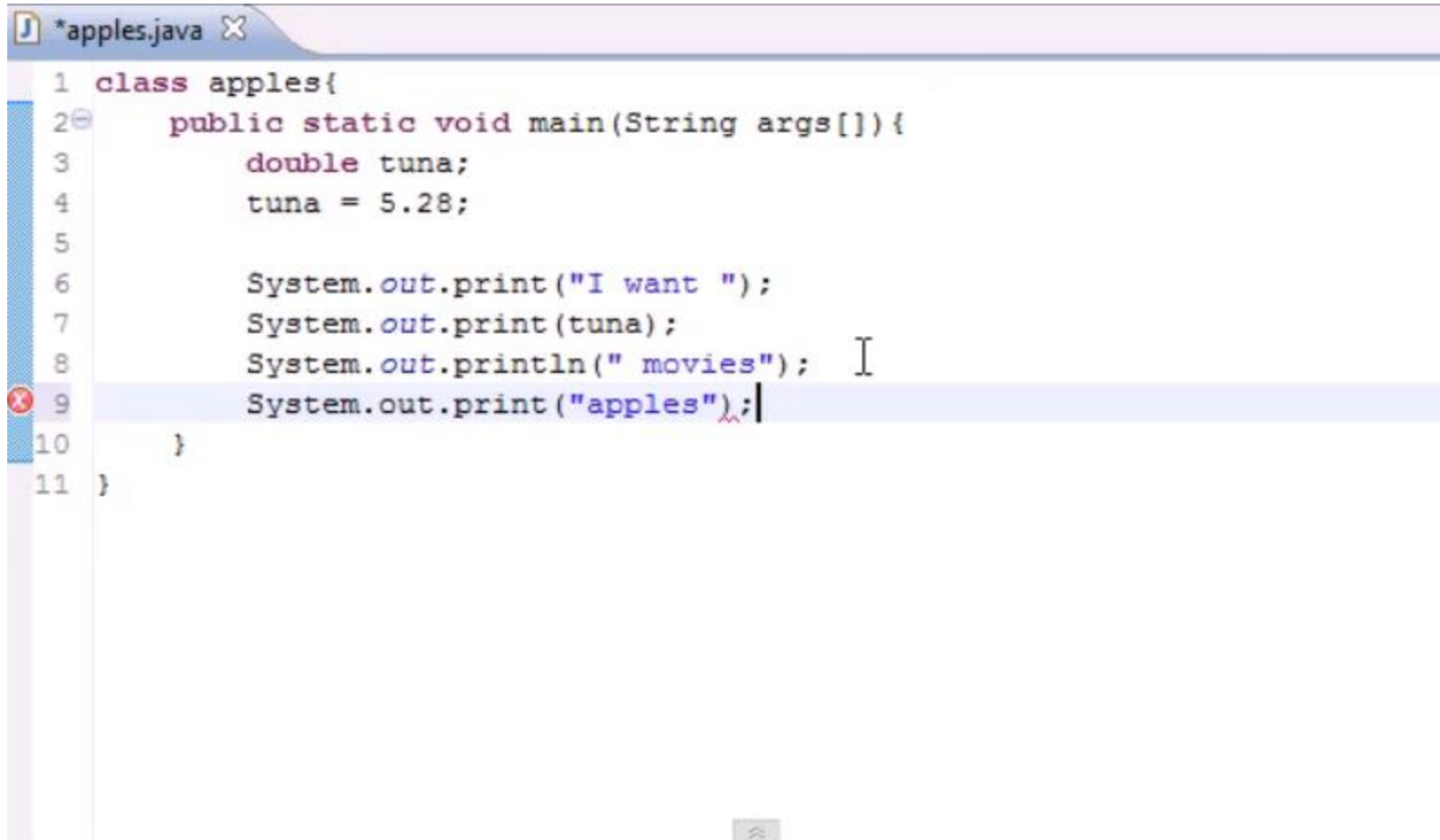
```
1 class apples{
2     public static void main(String args[]){
3         double tuna;
4         tuna = 5.28;
5
6         System.out.print("I want ");
7         System.out.print(tuna);
8         System.out.println(" movies");
9     }
10 }
```



# View output changes

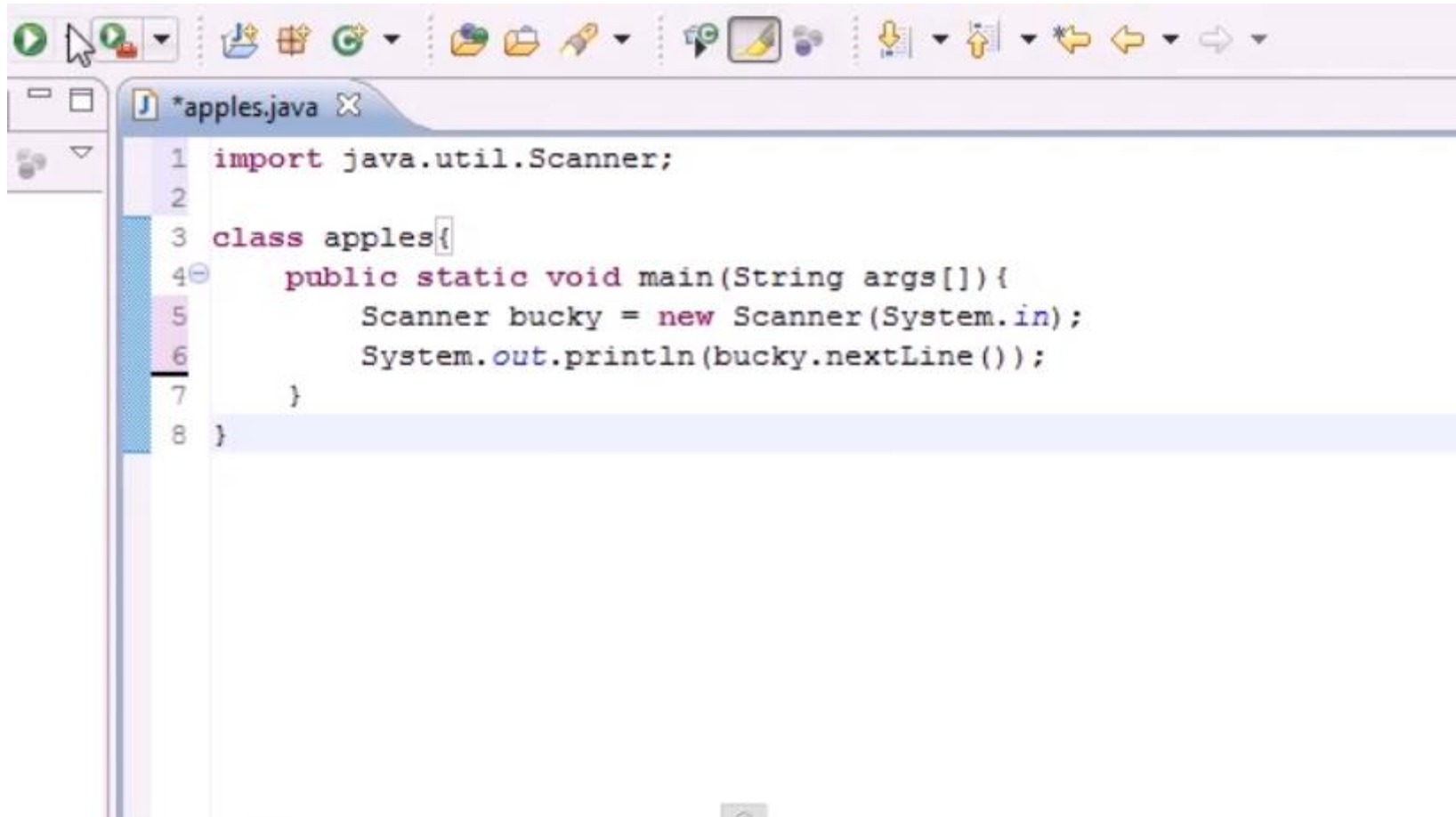


# Add another print



```
1 class apples{
2     public static void main(String args[]){
3         double tuna;
4         tuna = 5.28;
5
6         System.out.print("I want ");
7         System.out.print(tuna);
8         System.out.println(" movies");
9         System.out.print("apples");
10    }
11 }
```

# Getting user input in Java

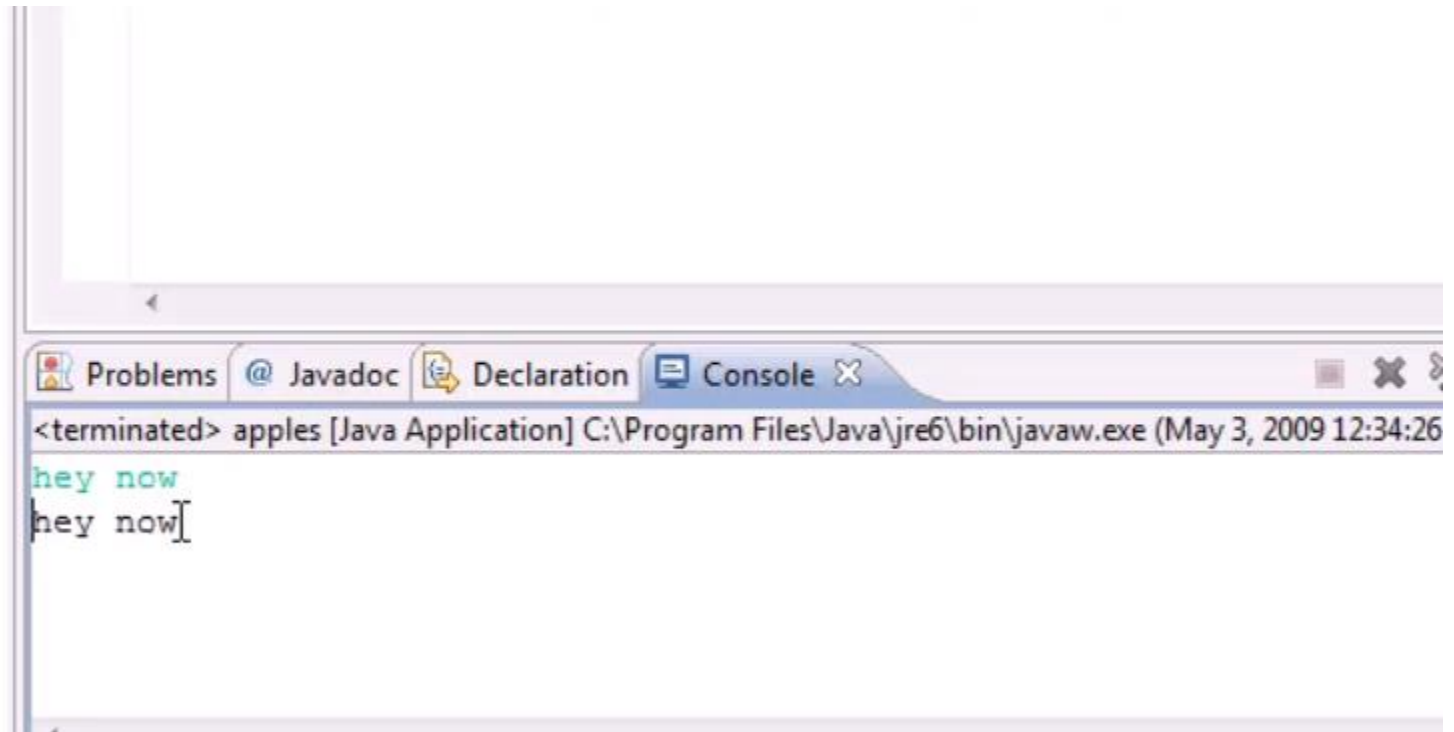


The screenshot shows an IDE window titled `*apples.java`. The code is as follows:

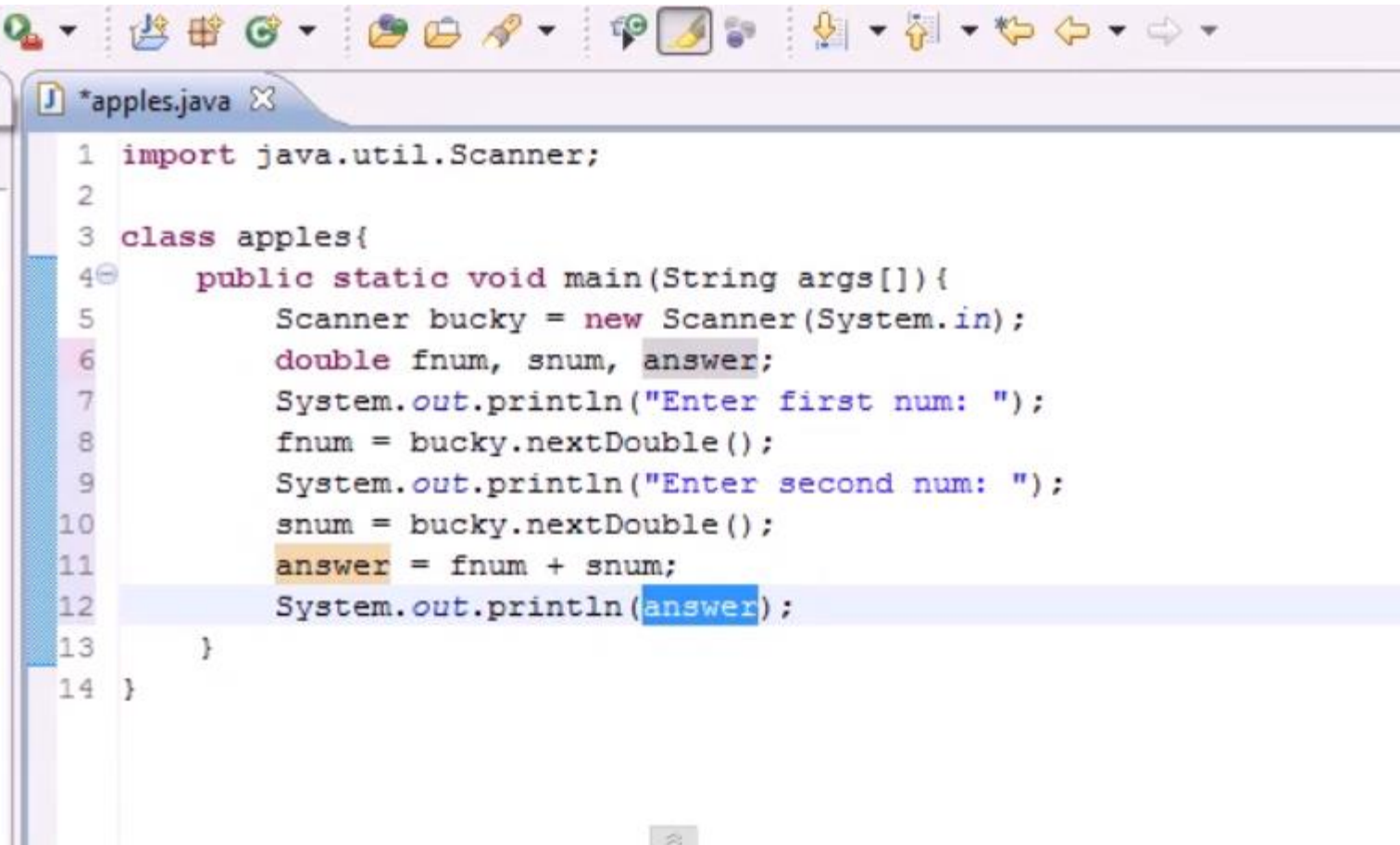
```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6         System.out.println(bucky.nextLine());
7     }
8 }
```

The code imports the `java.util.Scanner` class, defines a class named `apples`, and implements a `main` method. Inside the `main` method, a `Scanner` object named `bucky` is created using `System.in` as the source. The `bucky.nextLine()` method is then called to read a line of input from the user, and the result is printed to the console using `System.out.println`.

# View output



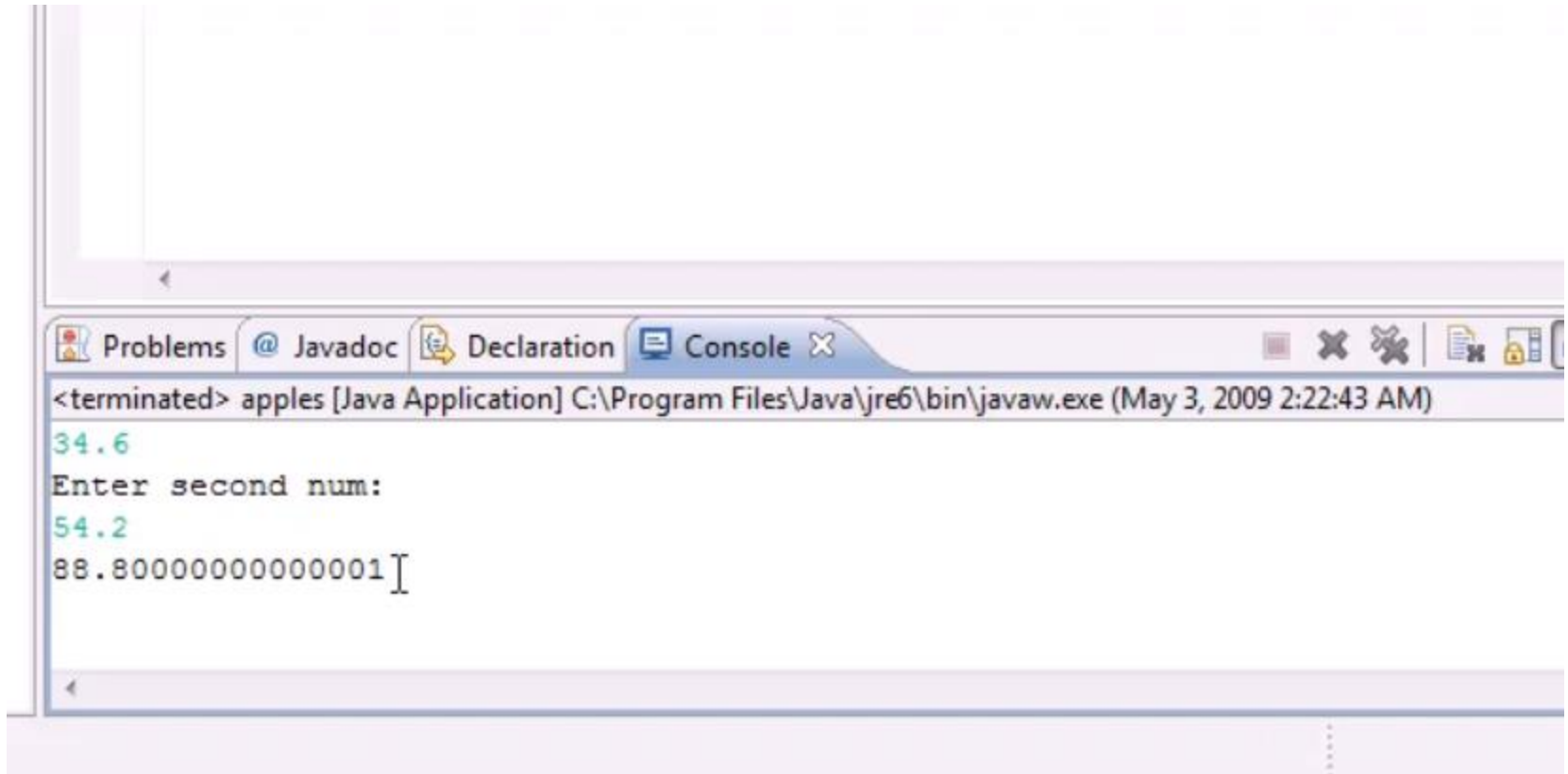
# A basic calculator in Java



The image shows a screenshot of an IDE window titled '\*apples.java'. The code is a Java program that implements a basic calculator. It imports the Scanner class and defines a class named 'apples' with a main method. The main method uses a Scanner to read two double values from the user, adds them, and prints the result. The code is as follows:

```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6         double fnum, snum, answer;
7         System.out.println("Enter first num: ");
8         fnum = bucky.nextDouble();
9         System.out.println("Enter second num: ");
10        snum = bucky.nextDouble();
11        answer = fnum + snum;
12        System.out.println(answer);
13    }
14 }
```

# View output



# Java Language Basics

## Operators

- Operators are special symbols that perform specific operations on one, two, or three *operands*, and then return a result.

# Java Language Basics

## Operators

### The Arithmetic Operators

- + additive operator, e.g, 3 + 9
- - subtraction operator, e.g, 3 - 9
- \* multiplication operator, e.g, 3 \* 9
- / division operator, e.g, 9 / 3
- % remainder operator, e.g, 9 % 3



# Java Language Basics

## Operators

### The Simple Assignment Operator

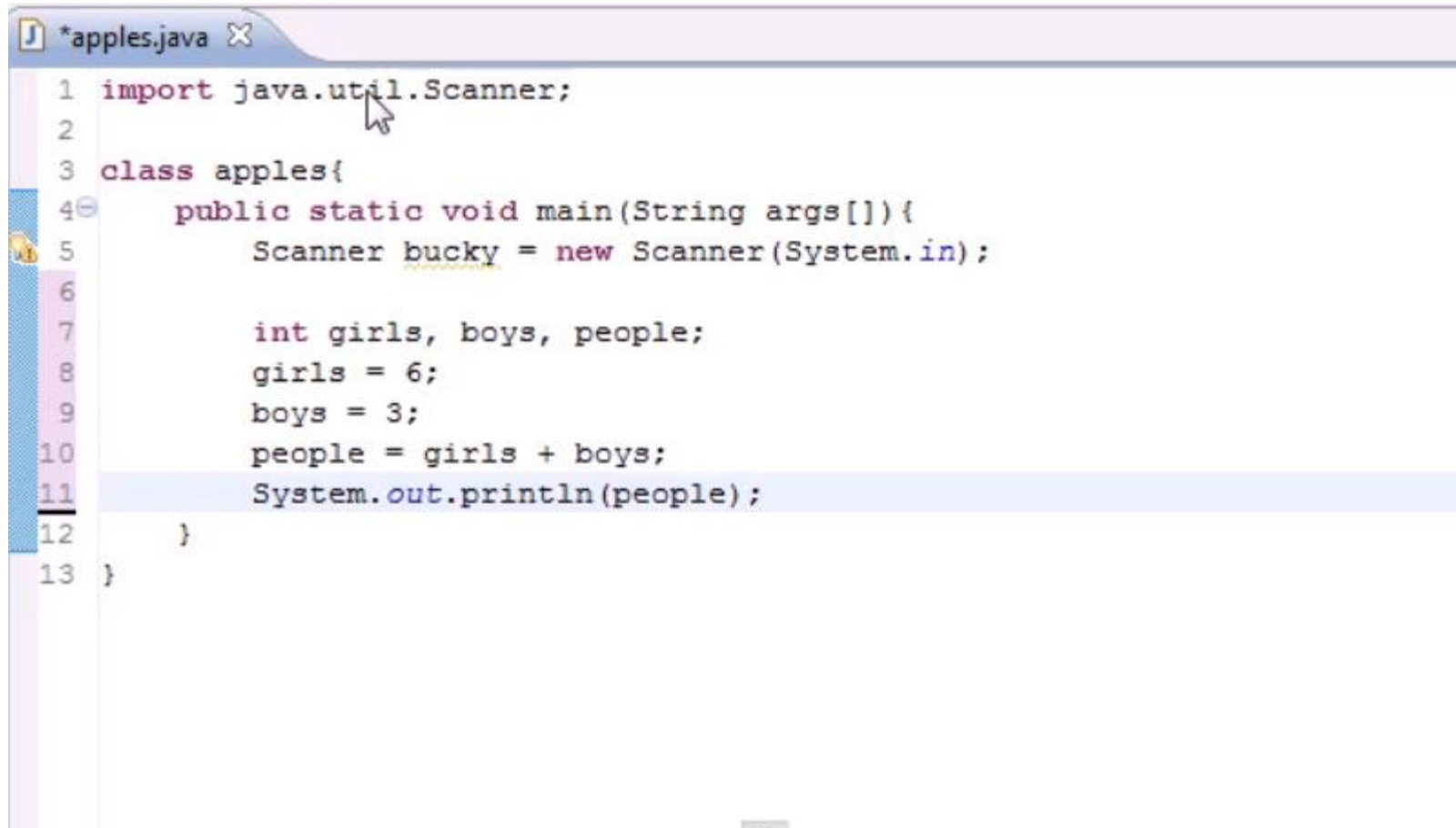
- One of the most common operators that you'll encounter is the simple assignment operator "`=`".
- it assigns the value on its right to the operand on its left:
  - `int cadence = 0;`
  - `int speed = 0;`
  - `int gear = 1;`

# Java Language Basics

## Operators

- **The Increment/Decrement Operator**
- ++ Increments value by one
  - `int speed = 0;`
  - `++x`
- -- Decrements value by one
  - `int speed = 0;`
  - `--x`

# Basic math operators in Java: plus



The screenshot shows a Java IDE window titled '\*apples.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 6;
9         boys = 3;
10        people = girls + boys;
11        System.out.println(people);
12    }
13 }
```

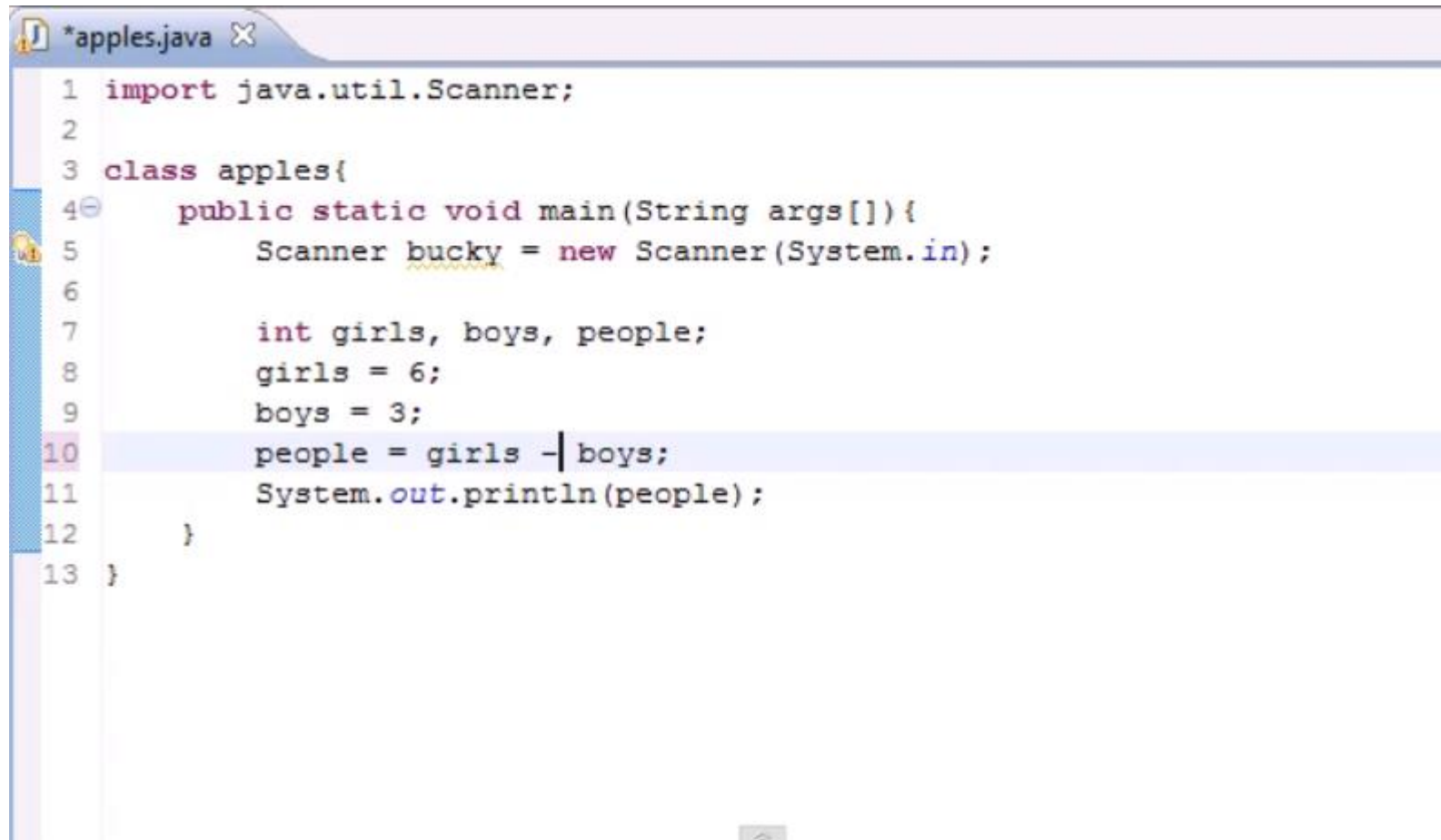
The code demonstrates the use of the plus operator (+) to calculate the sum of 'girls' and 'boys' and store the result in 'people'. The line 'people = girls + boys;' is highlighted in blue.

# View output



9 I

# Basic math operators in Java: minus



The screenshot shows a Java IDE window titled "\*apples.java". The code is as follows:

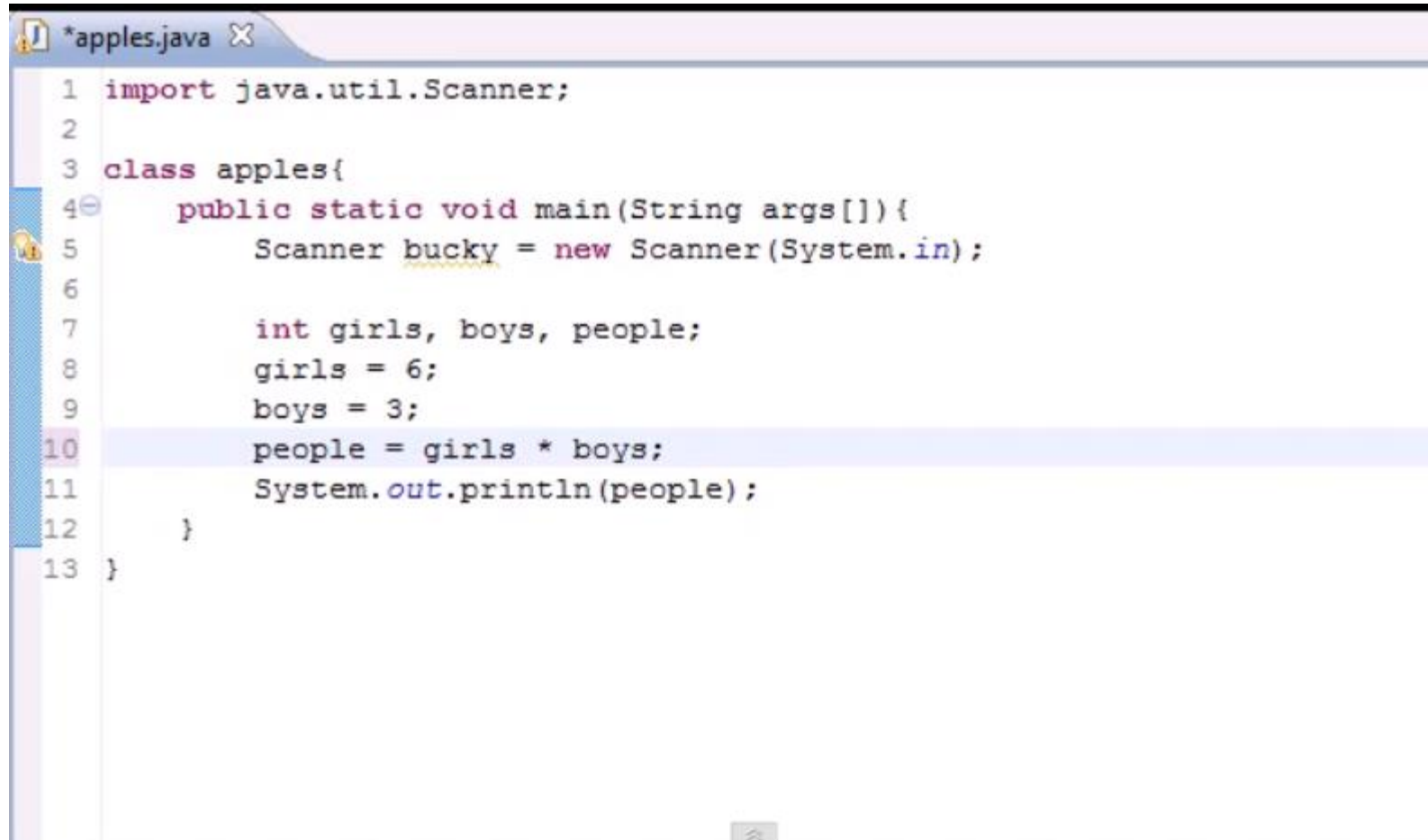
```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 6;
9         boys = 3;
10        people = girls - boys;
11        System.out.println(people);
12    }
13 }
```

The line `people = girls - boys;` on line 10 is highlighted, demonstrating the use of the minus operator.

# View output



# Basic math operators in Java: multiply

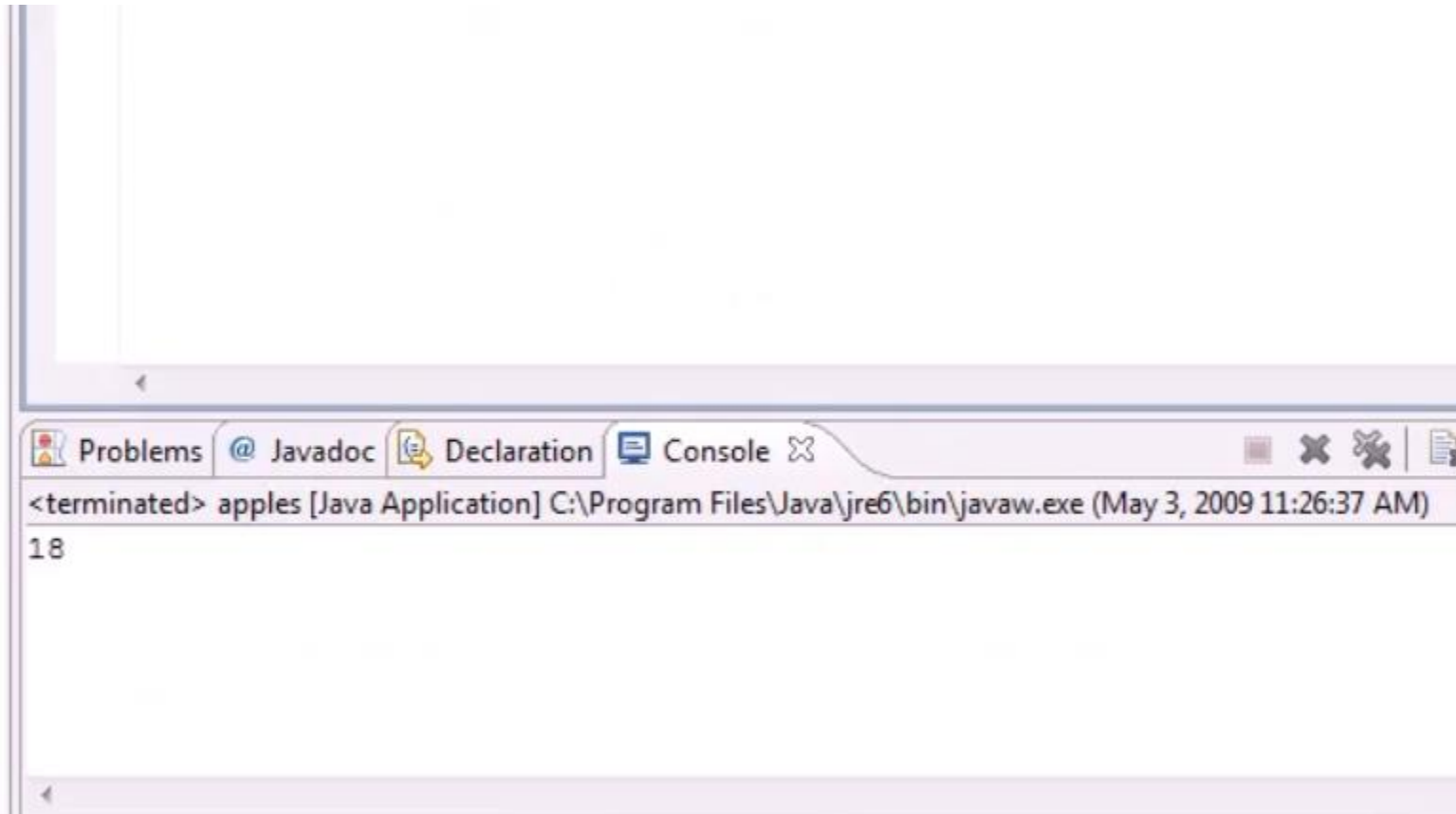


The screenshot shows a Java IDE window titled '\*apples.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 6;
9         boys = 3;
10        people = girls * boys;
11        System.out.println(people);
12    }
13 }
```

The line `people = girls * boys;` is highlighted in blue. A small icon of a person is visible in the left margin next to line 5.

# View output





# Basic math operators in Java: division

```
*apples.java X
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 6;
9         boys = 3;
10        people = girls / boys;
11        System.out.println(people);
12    }
13 }
```

# Basic math operators in Java: division

```
*apples.java X
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 11;
9         boys = 3;
10        people = girls / boys;
11        System.out.println(people);
12    }
13 }
```

# View output



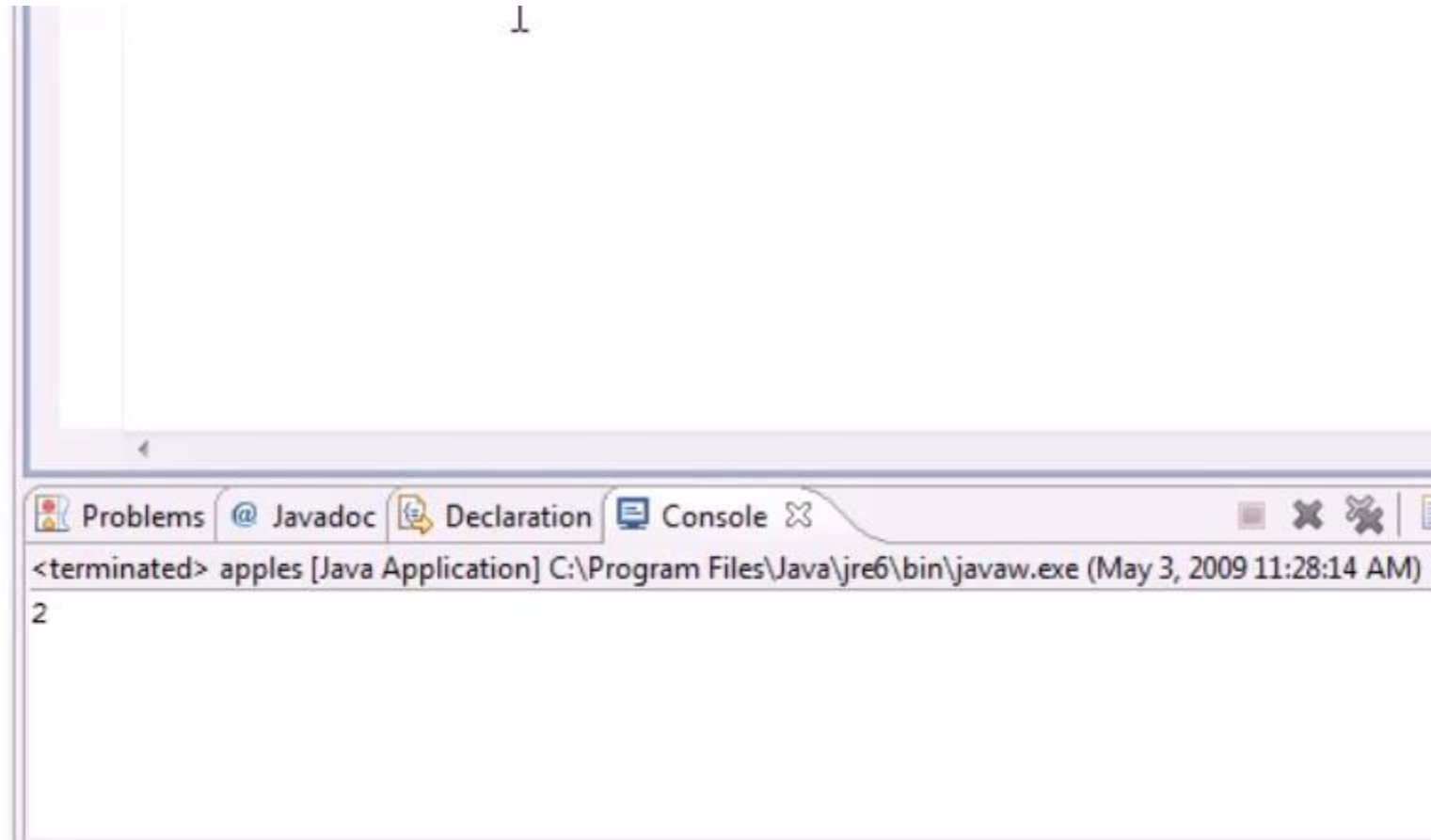
# Basic math operators in Java

```
apples.java ✕
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         double girls, boys, people;
8         girls = 11;
9         boys = 3;
10        people = girls / boys;
11        System.out.println(people);
12    }
13 }
```

# Basic math operators in Java: remainder

```
*apples.java X
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int girls, boys, people;
8         girls = 11;
9         boys = 3;
10        people = girls % boys;
11        System.out.println(people);
12    }
13 }
```

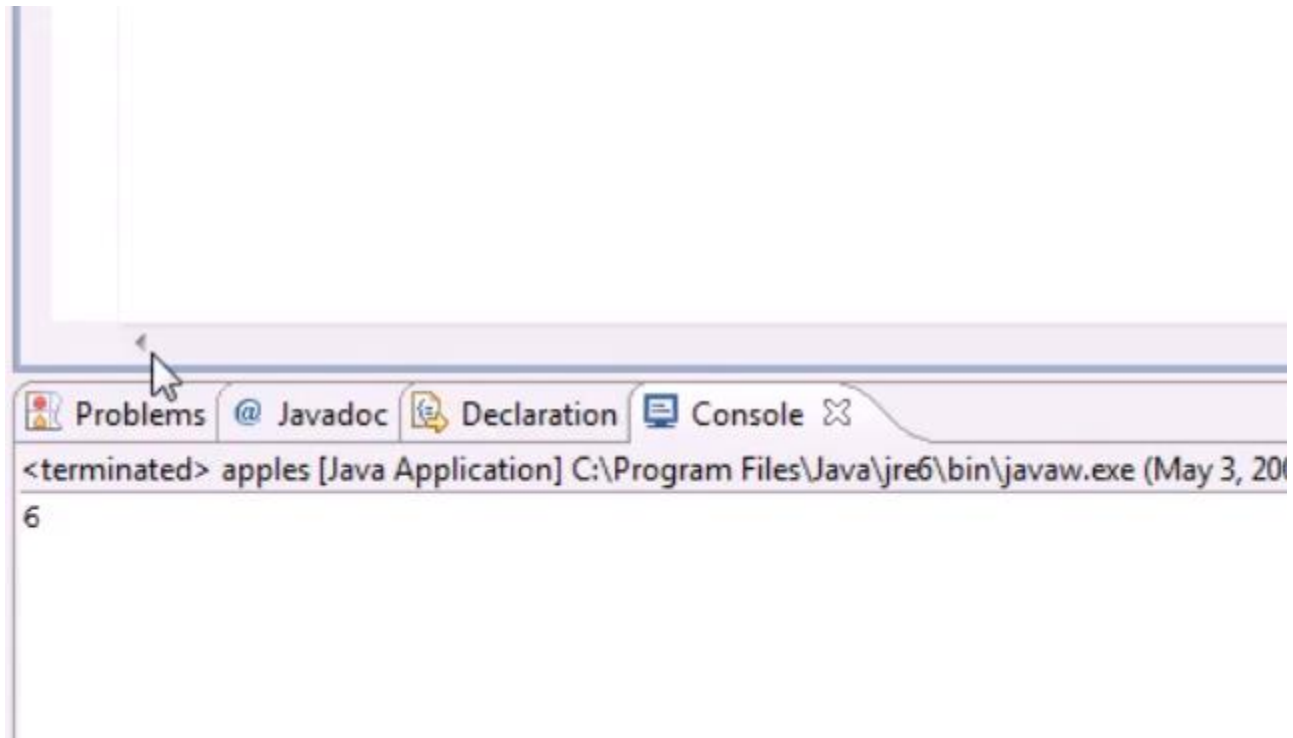
# View output



# Incr/Decr operators in Java

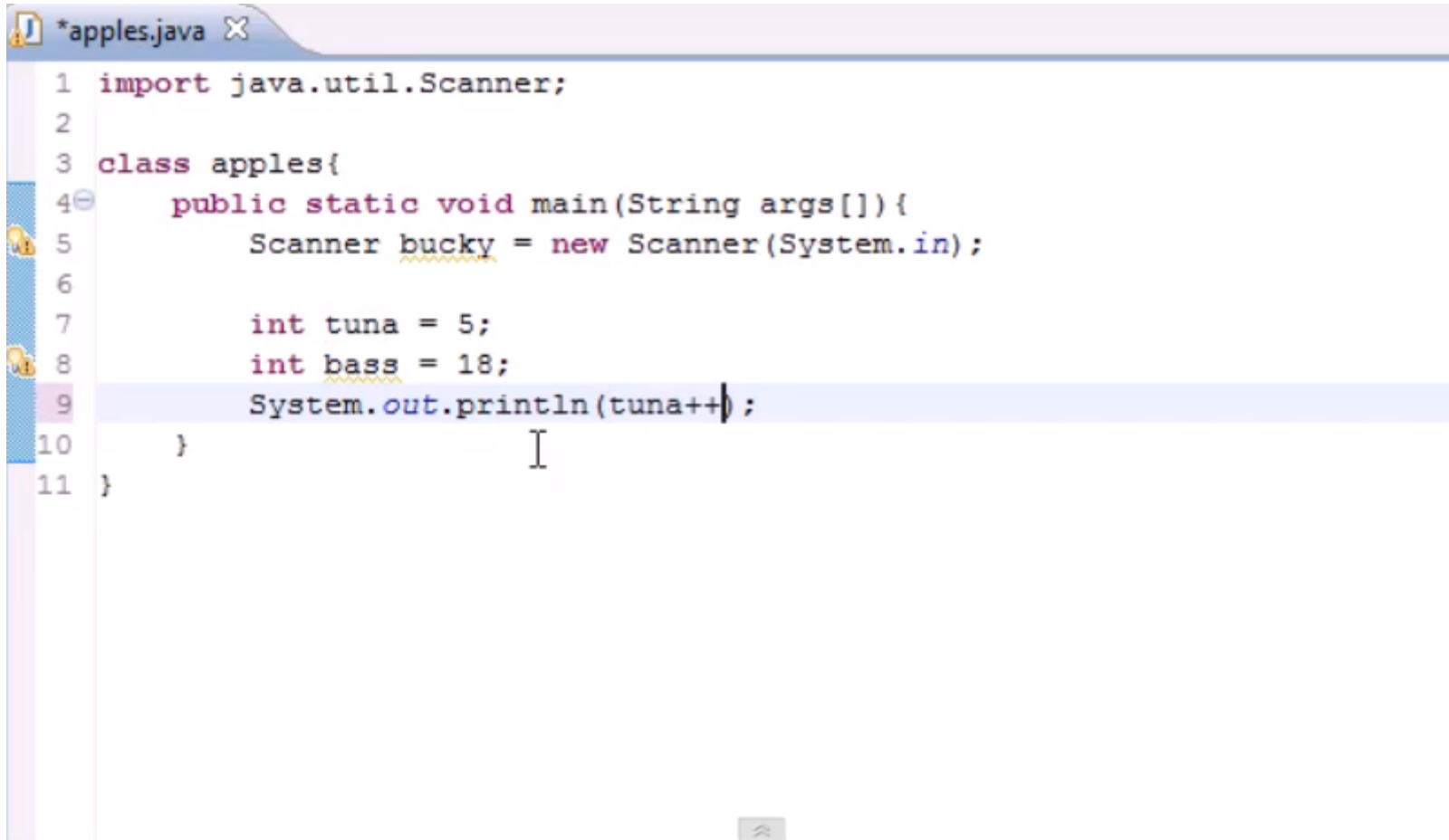
```
*apples.java X
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9         ++tuna;
10        System.out.println(tuna);
11    }
12 }
```

# View output





# Post increment operator

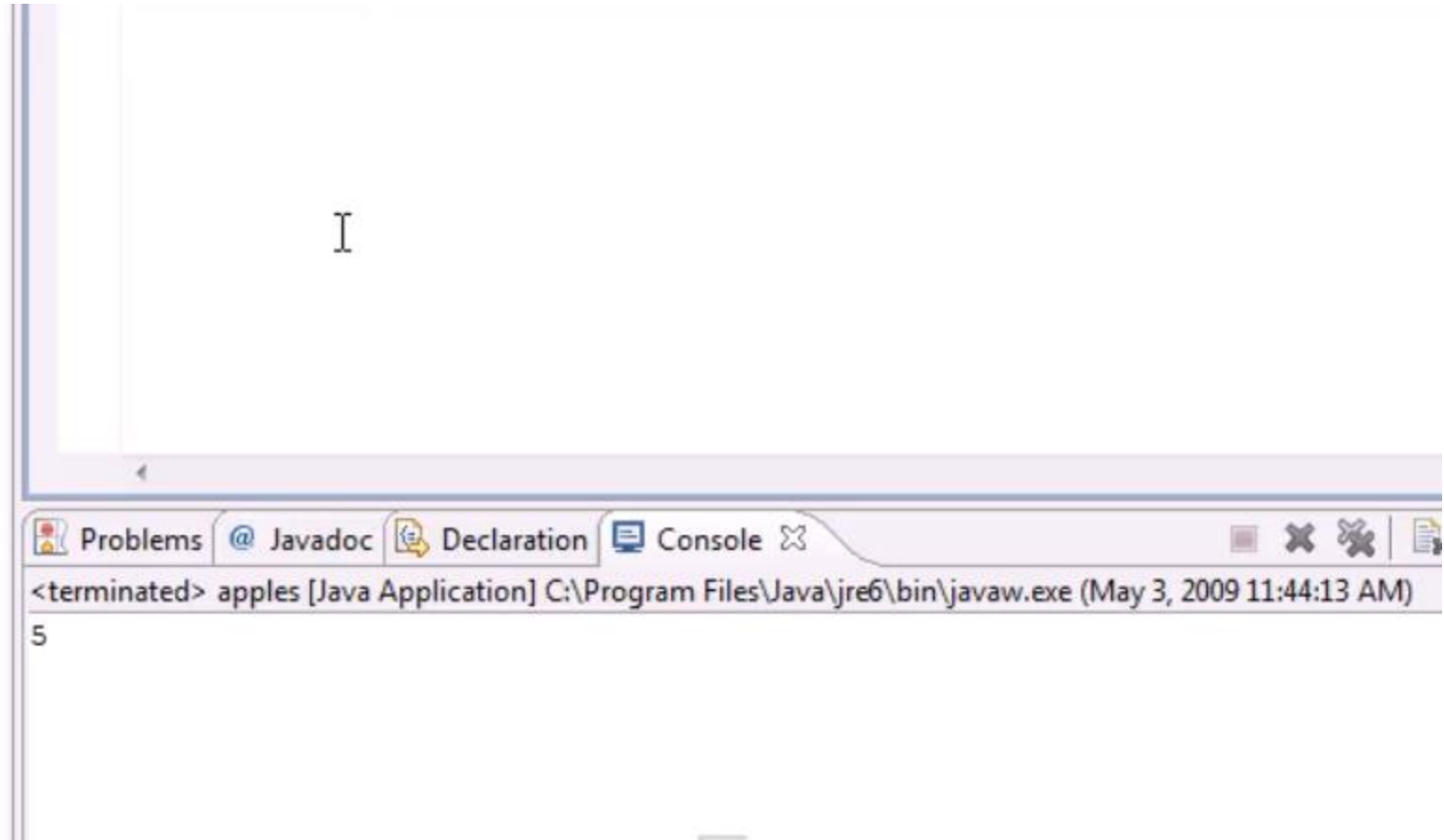


The screenshot shows a Java IDE window titled `*apples.java`. The code is as follows:

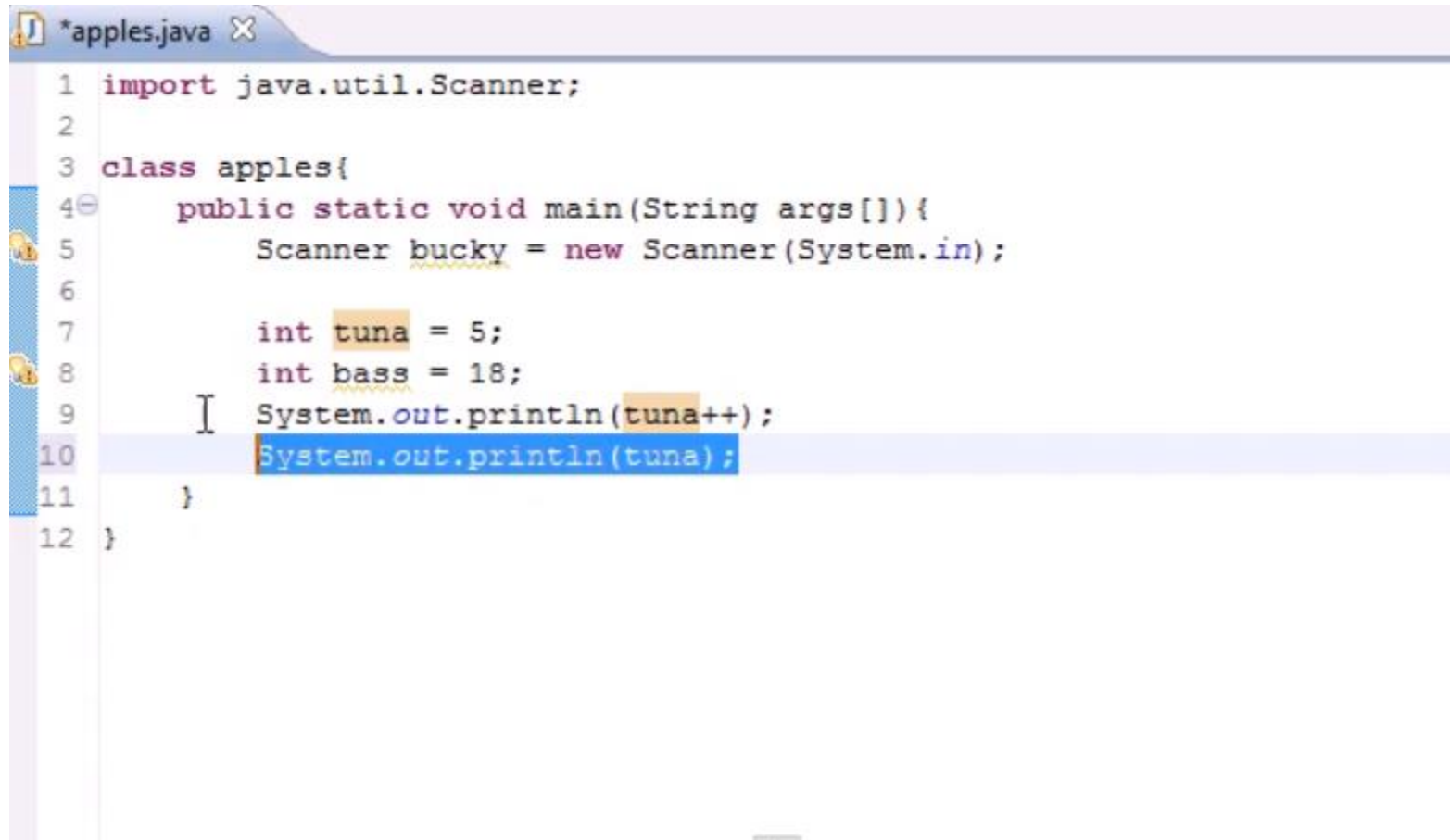
```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9         System.out.println(tuna++);
10    }
11 }
```

The line `System.out.println(tuna++);` is highlighted in blue. A cursor is positioned at the end of this line. The variable `tuna` is initialized to 5, and the `++` operator is used to increment its value before printing.

# View output

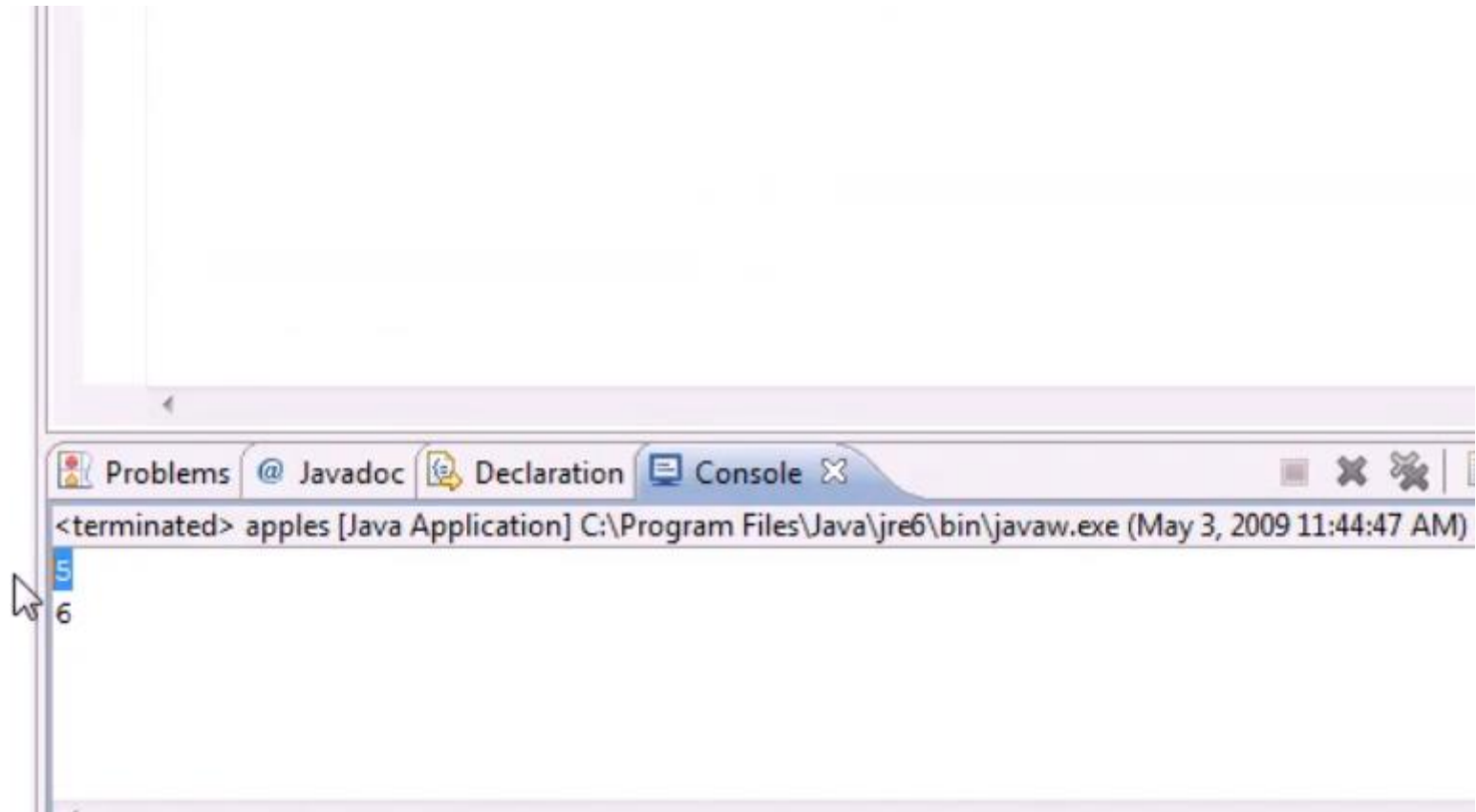


# Post increment operator (more)

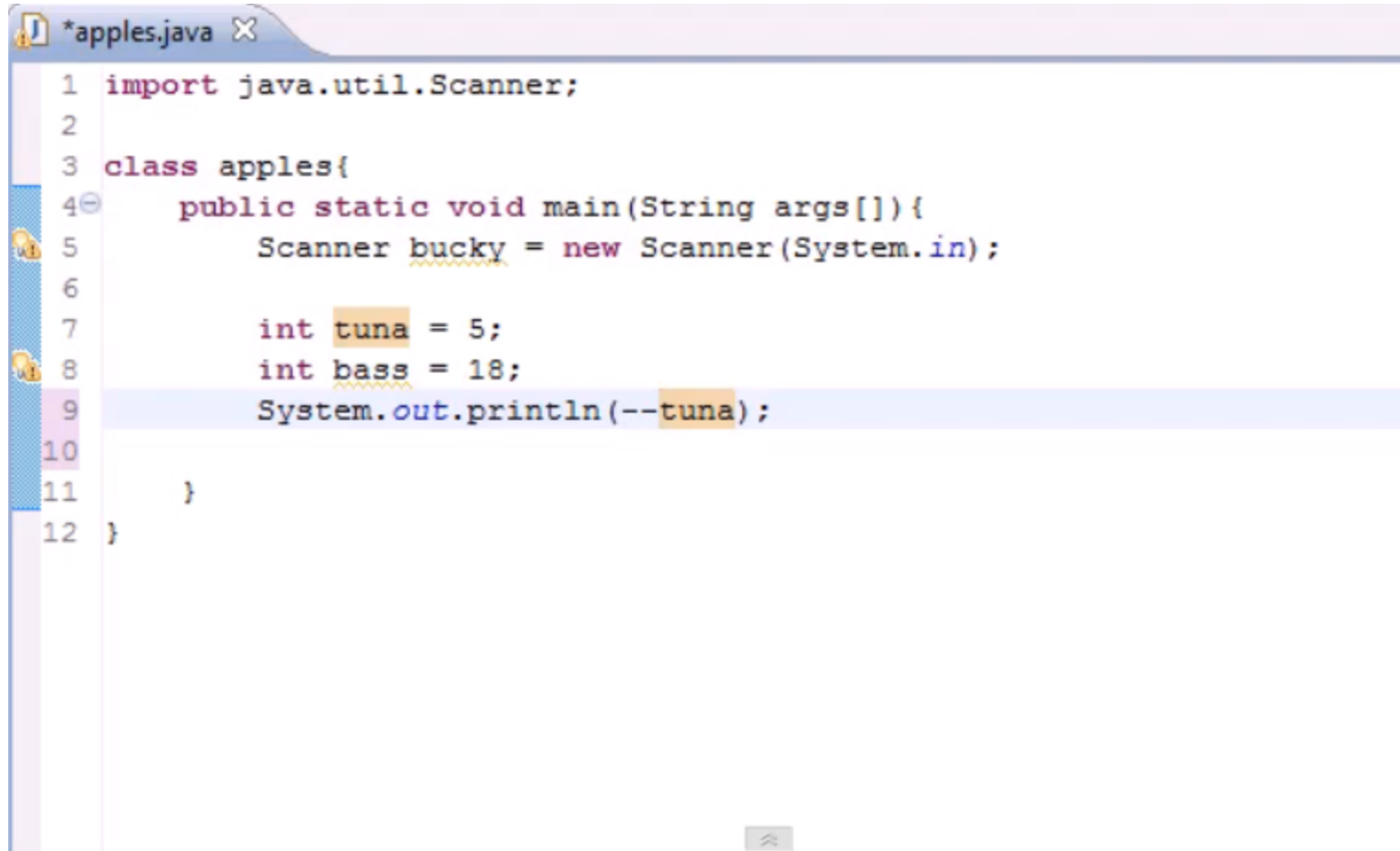


```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9         System.out.println(tuna++);
10        System.out.println(tuna);
11    }
12 }
```

# View output

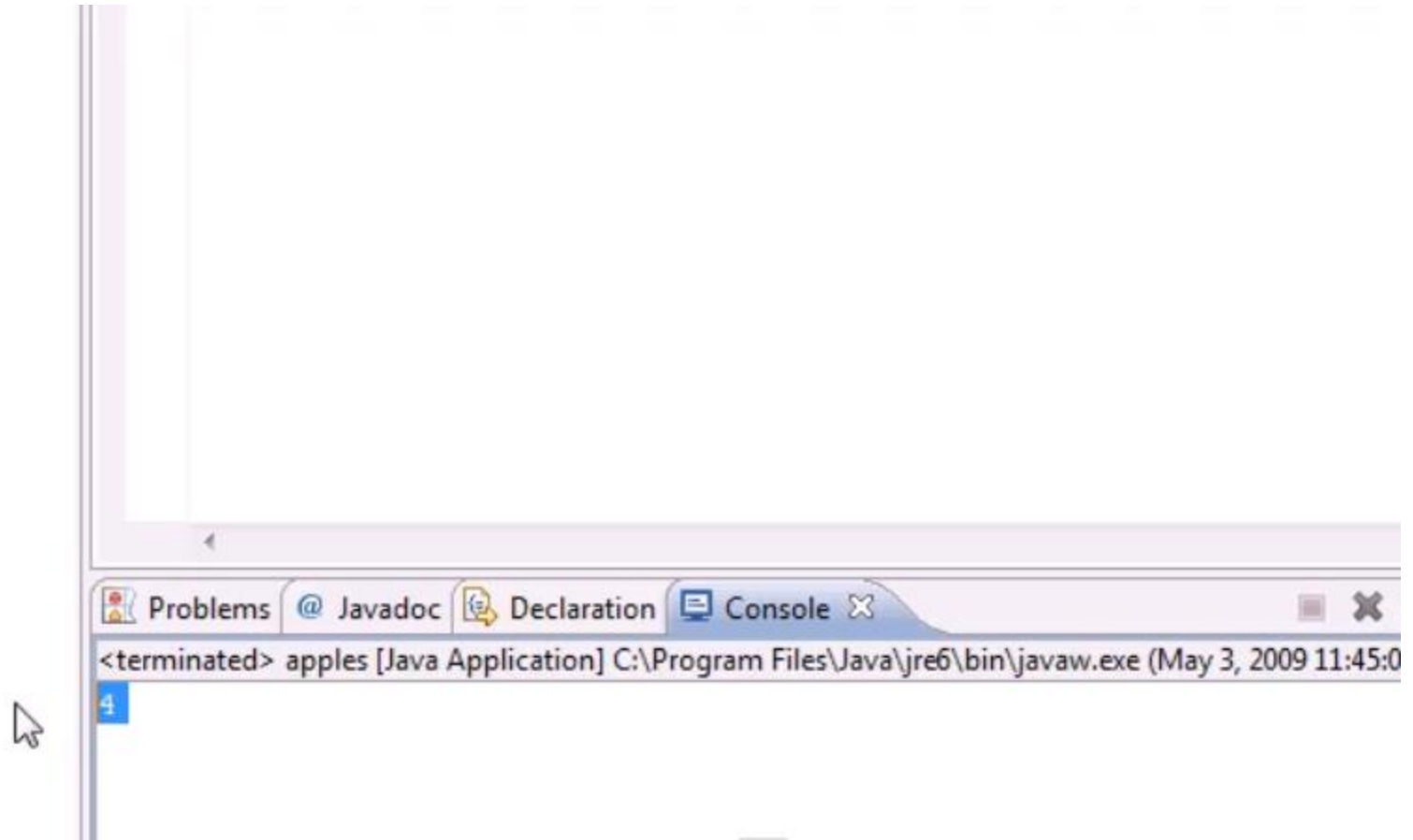


# Pre decrement operator

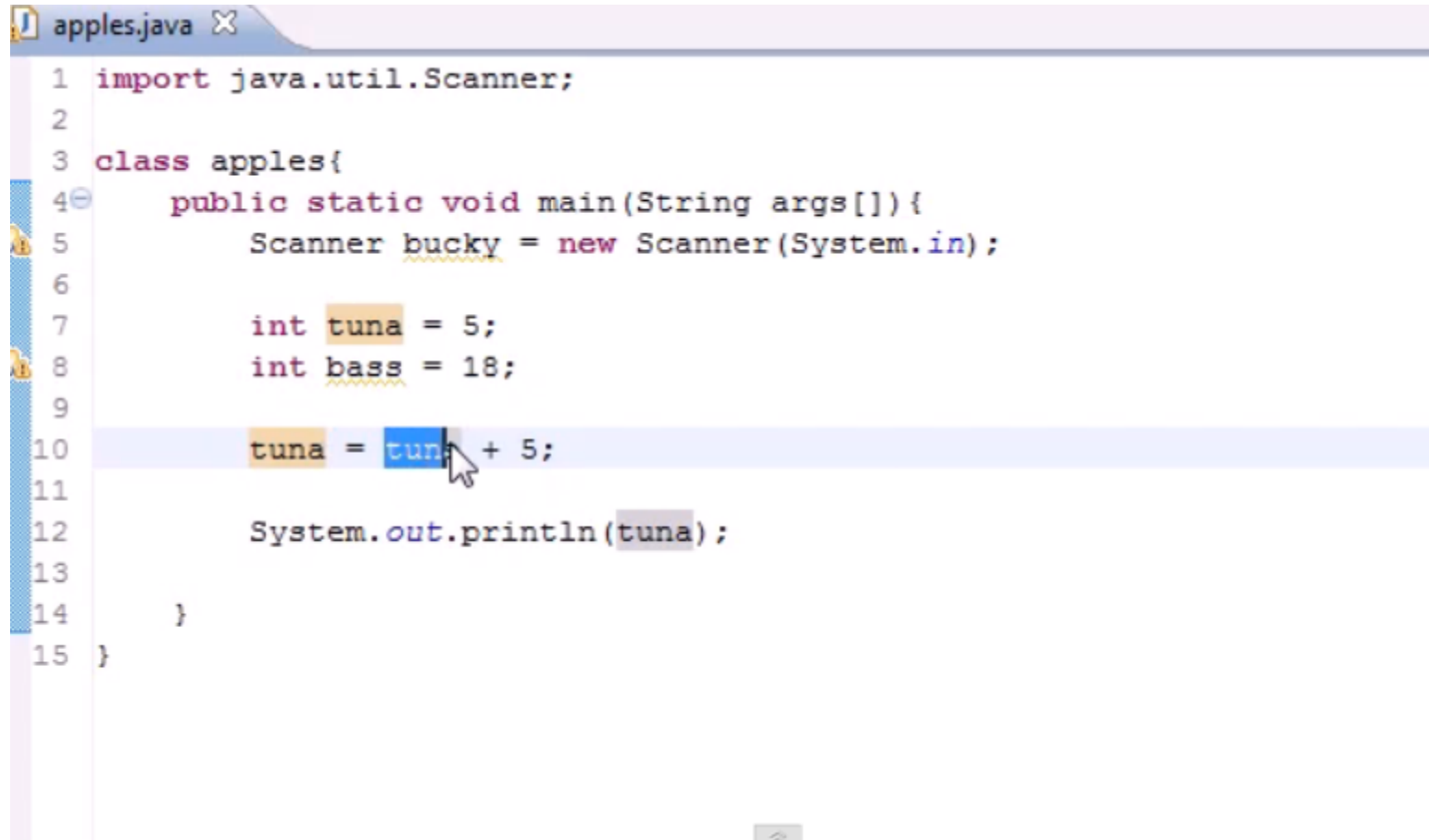


```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9         System.out.println(--tuna);
10
11     }
12 }
```

# View output



# Assignment operator in Java

A screenshot of a Java IDE window titled 'apples.java'. The code defines a class 'apples' with a 'main' method. Inside the 'main' method, a 'Scanner' object named 'bucky' is created. Two integer variables, 'tuna' and 'bass', are declared and initialized. The variable 'tuna' is then updated using the assignment operator '='. The line 'tuna = tuna + 5;' is highlighted in blue, and a mouse cursor is pointing at the second 'tuna' operand. Finally, the value of 'tuna' is printed to the console.

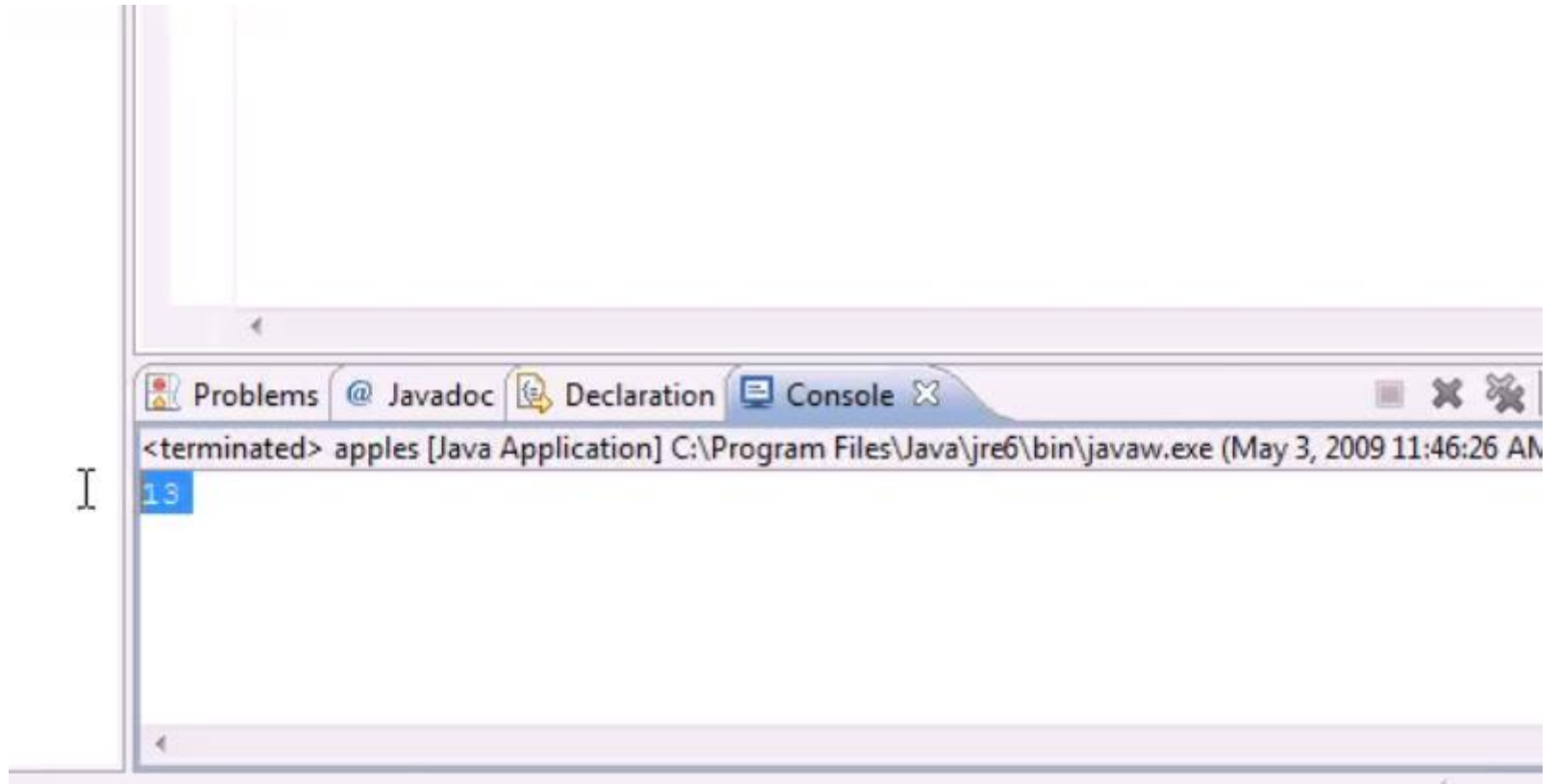
```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9
10        tuna = tuna + 5;
11
12        System.out.println(tuna);
13
14    }
15 }
```

# Composite assignment operator: +=

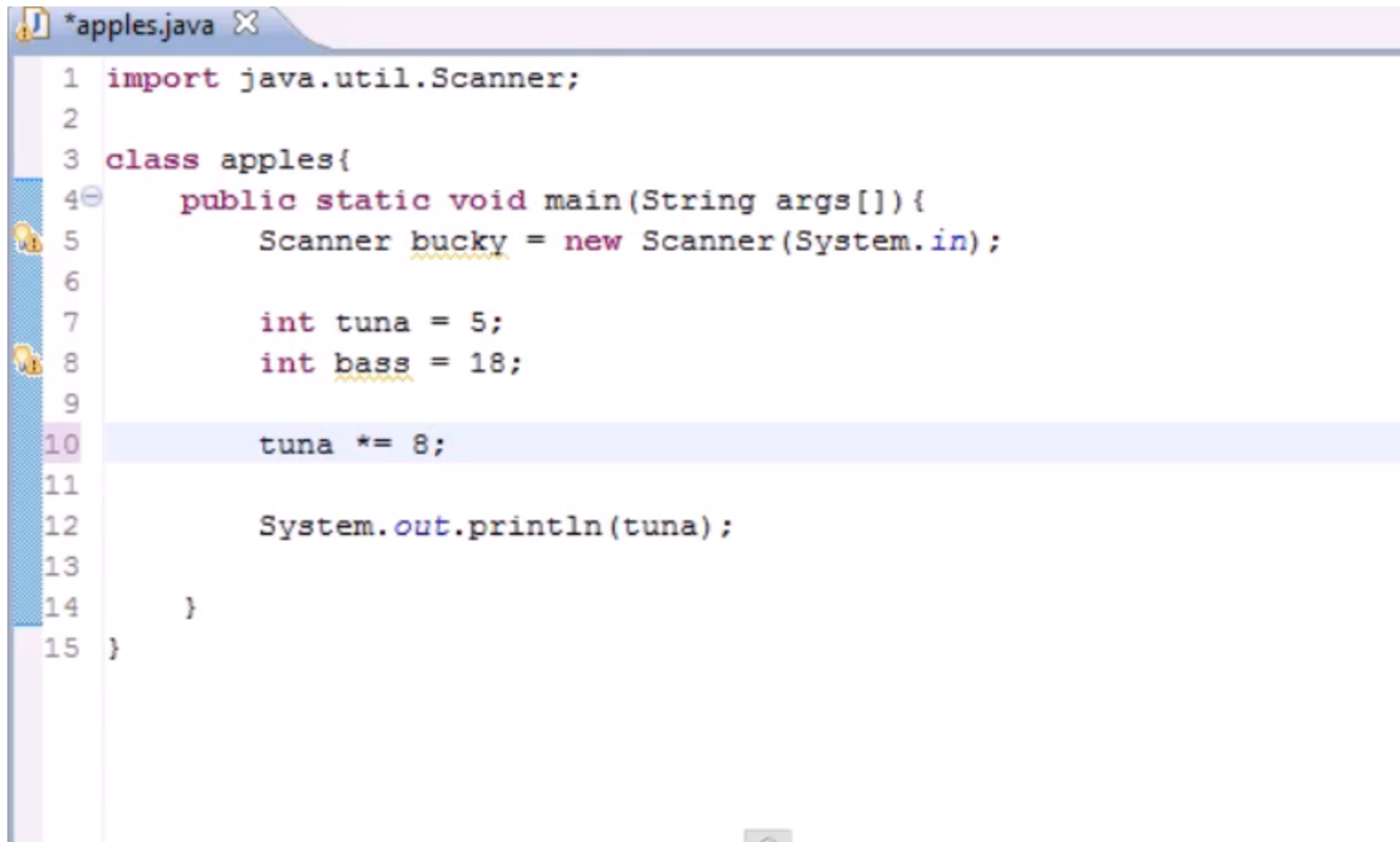
```
*apples.java X
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9
10        tuna += 8;
11
12        System.out.println(tuna);
13
14    }
15 }
```



# View output

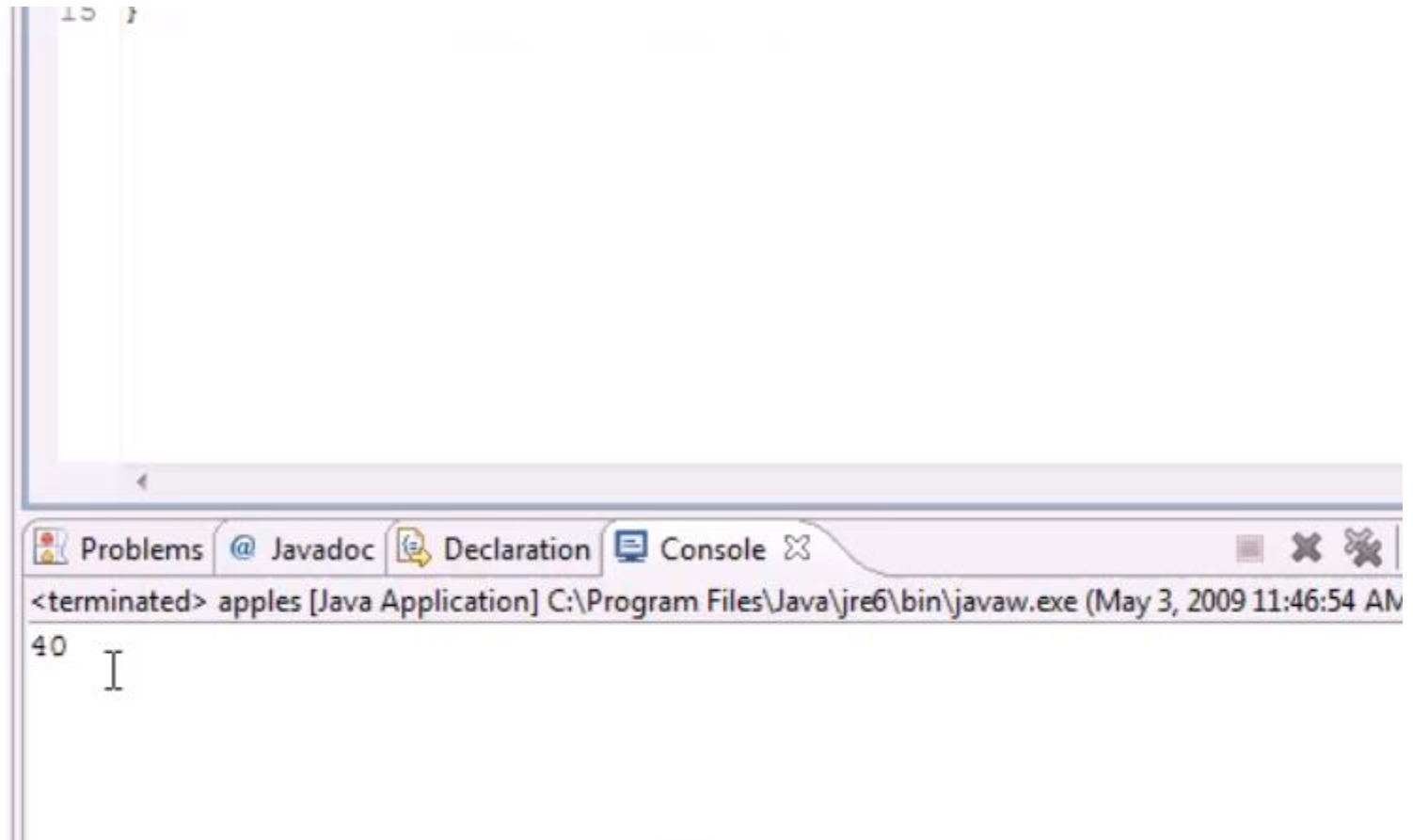


# Composite assignment operators: \*=



```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String args[]){
5         Scanner bucky = new Scanner(System.in);
6
7         int tuna = 5;
8         int bass = 18;
9
10        tuna *= 8;
11
12        System.out.println(tuna);
13
14    }
15 }
```

# View output



# Control Flow Statements

- The statements inside your source files are **generally executed from top to bottom**, in the order that they appear.
- *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code.

# Control Flow Statements

Control flow statements include-

- **decision-making statements** (if-then, if-then-else, switch)
- **the looping statements** (for, while, do-while),
- **the branching statements** (break, continue, return)

# The if-then and if-then-else Statements

## The if-then Statement

- It tells your program to **execute** a certain section of **code *only if*** a particular test evaluates to true.

```
void applyBrakes() {  
    // the "if" clause: bicycle must be moving  
    if (isMoving){  
        // the "then" clause: decrease current speed  
        currentSpeed--;  
    }  
}
```

# The if-then and if-then-else Statements

## The if-then-else Statement

- The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.

```
void applyBrakes() {  
    if (isMoving) {  
        currentSpeed--;  
    } else {  
        System.err.println("The bicycle has " + "already stopped!");  
    }  
}
```

# The if-then and if-then-else Statements

## The if-then-else Statement

- The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.

```
void applyBrakes() {  
    if (isMoving) {  
        currentSpeed--;  
    } else {  
        System.err.println("The bicycle has " + "already stopped!");  
    }  
}
```



# The if-then and if-then-else Statements

## The switch Statement

- Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths.
- A switch works with the byte, short, char, and int primitive data types.

# The if-then and if-then-else Statements

## The Equality and Relational Operators

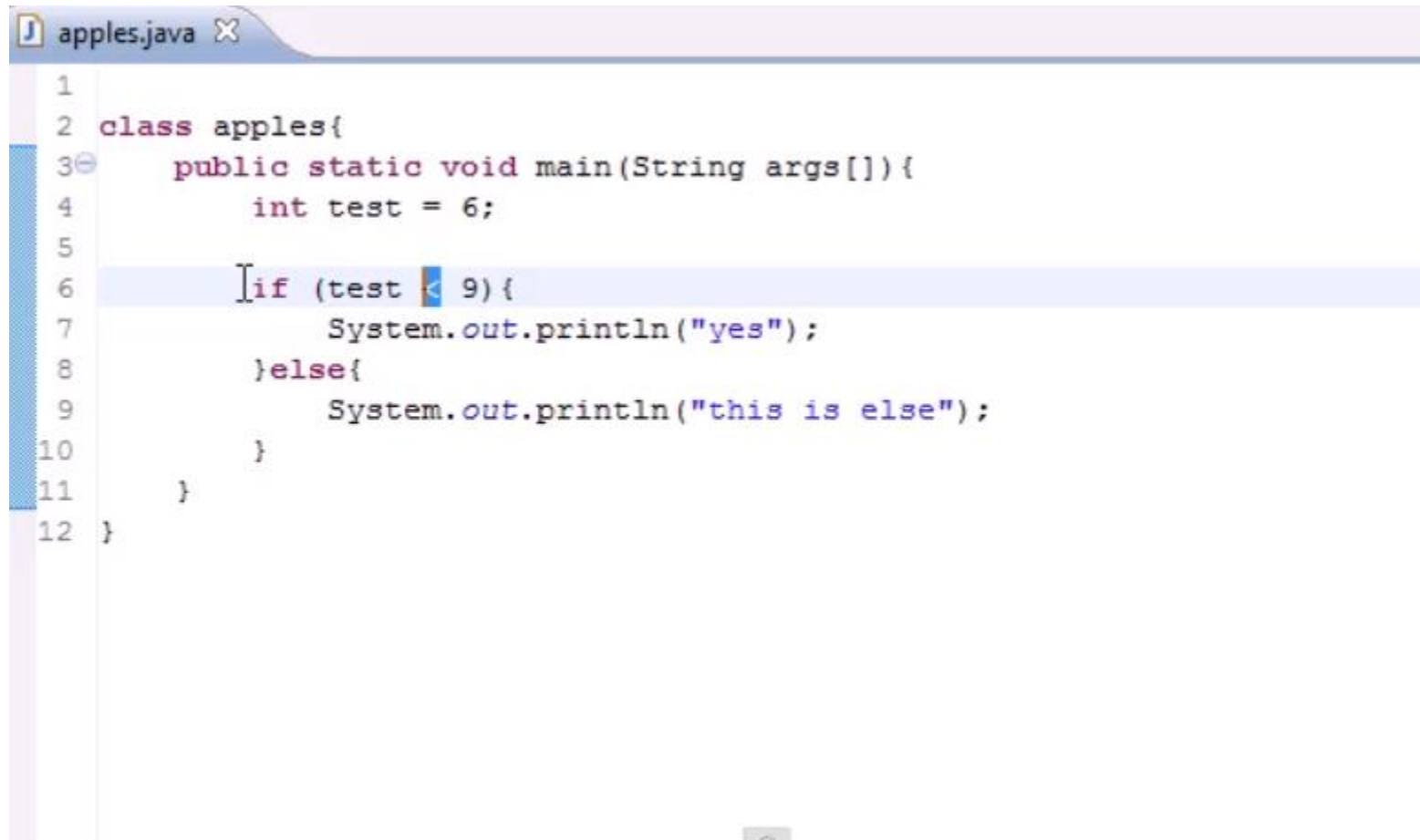
- The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.
- They are **used in testing conditions**.

<code>==</code>	<code>equal to</code>
<code>!=</code>	<code>not equal to</code>
<code>&gt;</code>	<code>greater than</code>
<code>&gt;=</code>	<code>greater than or equal to</code>
<code>&lt;</code>	<code>less than</code>
<code>&lt;=</code>	<code>less than or equal to</code>

# If-else statement in Java

```
*apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test == 9){
7             System.out.println("yes");
8         }else{
9             System.out.println("this is else");
10        }
11    }
12 }
```

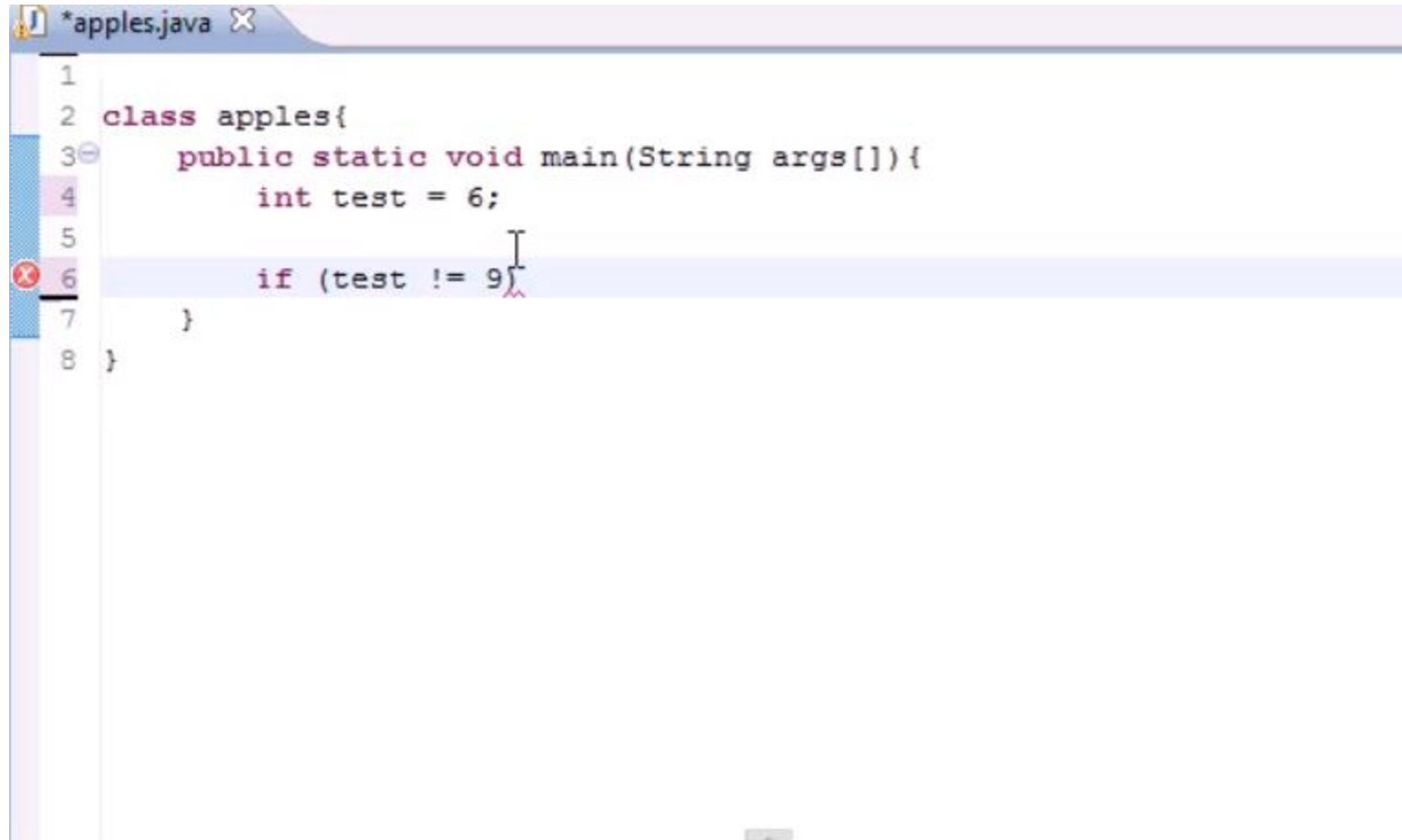
# If-else statement in Java



```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test < 9){
7             System.out.println("yes");
8         }else{
9             System.out.println("this is else");
10        }
11    }
12 }
```

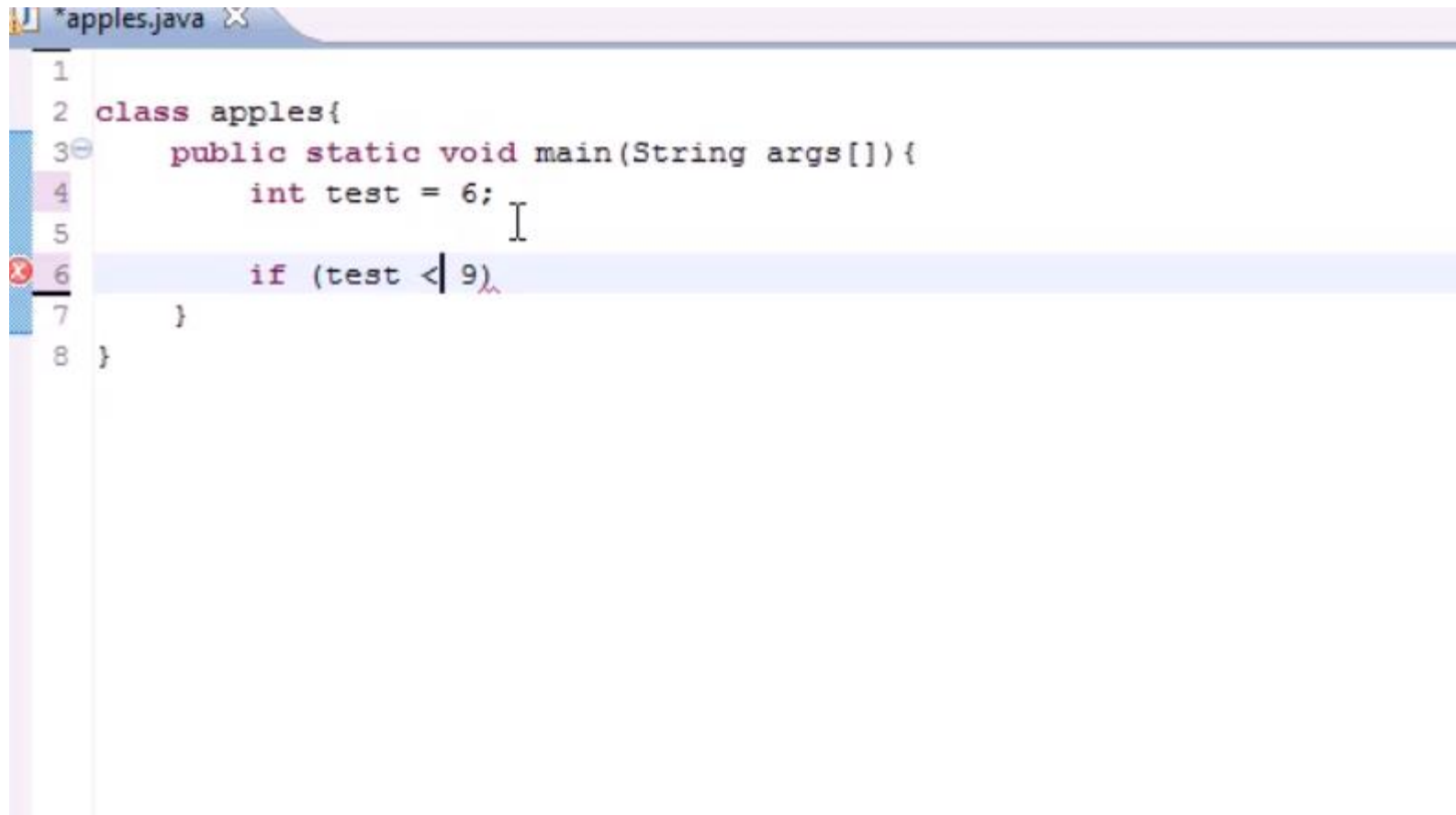
The screenshot shows a code editor window titled 'apples.java'. The code defines a class 'apples' with a 'main' method. Inside the 'main' method, an integer variable 'test' is initialized to 6. An 'if-else' statement follows: 'if (test < 9)' is true, so the first branch 'System.out.println("yes");' is executed. The second branch 'System.out.println("this is else");' is not executed. The code is properly indented and closed with curly braces.

# If-else statement in Java



```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test != 9){
7         }
8     }
```

# If-else statement in Java



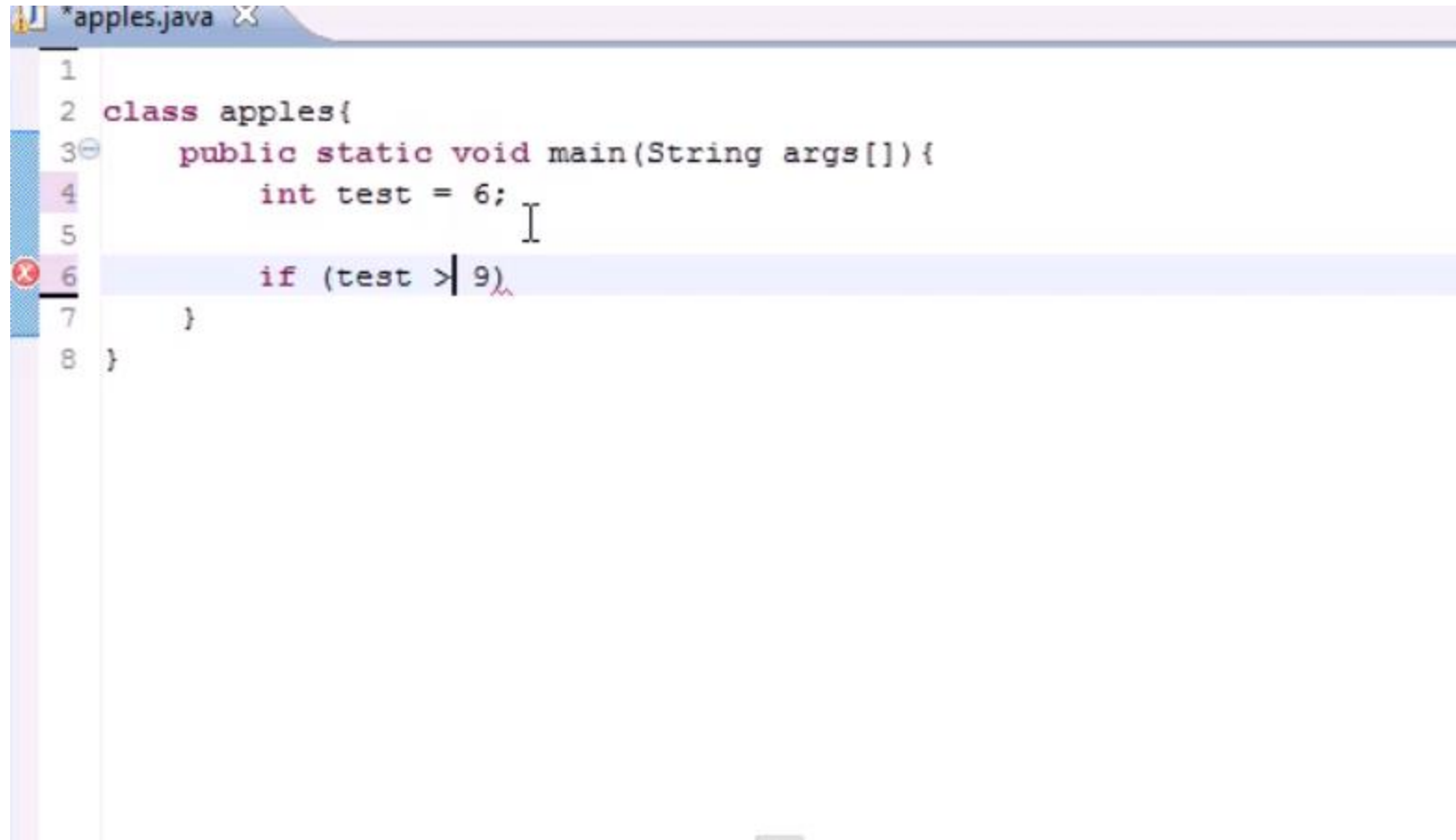
```
*apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test < 9){
7         }
8     }
```

The screenshot shows a code editor window titled '\*apples.java X'. The code is as follows:

```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test < 9){
7         }
8     }
```

The code defines a class named 'apples' with a 'main' method. Inside the 'main' method, an integer variable 'test' is initialized to 6. An 'if' statement is present, with the condition 'test < 9'. The body of the 'if' statement is currently empty, indicated by a closing curly brace on line 7. The cursor is positioned at the end of line 6, after the opening curly brace of the 'if' statement.

# If-else statement in Java

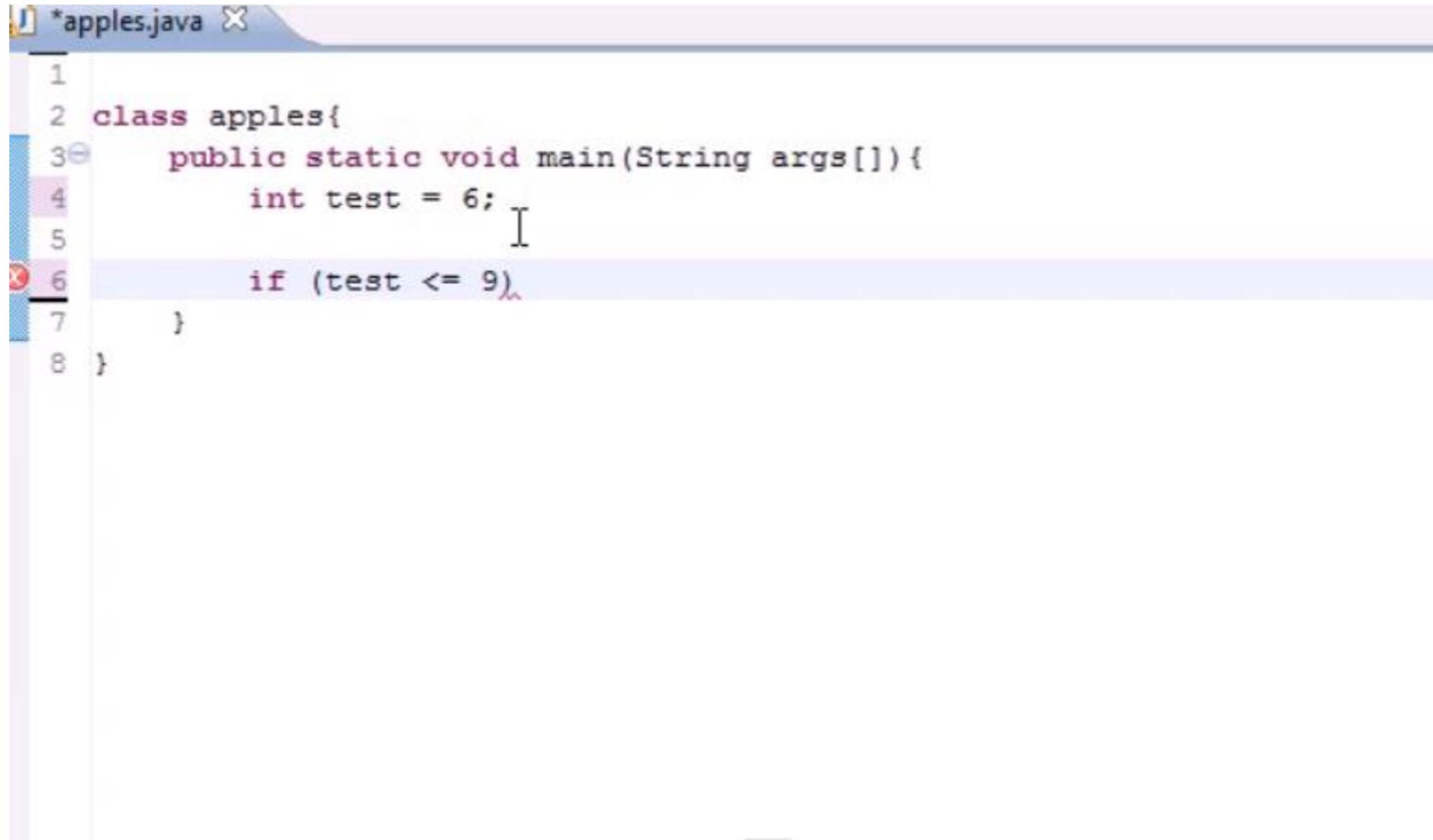


The screenshot shows a code editor window titled '\*apples.java'. The code is as follows:

```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test > 9)
7     }
8 }
```

The code defines a class named `apples` with a `main` method. Inside the `main` method, an integer variable `test` is initialized to 6. An `if` statement is present, with the condition `test > 9`. The `if` statement is currently empty, and the cursor is positioned at the end of the line `if (test > 9)`. The code is syntactically correct, but the condition `test > 9` will always be false given the initial value of `test`.

# If-else statement in Java



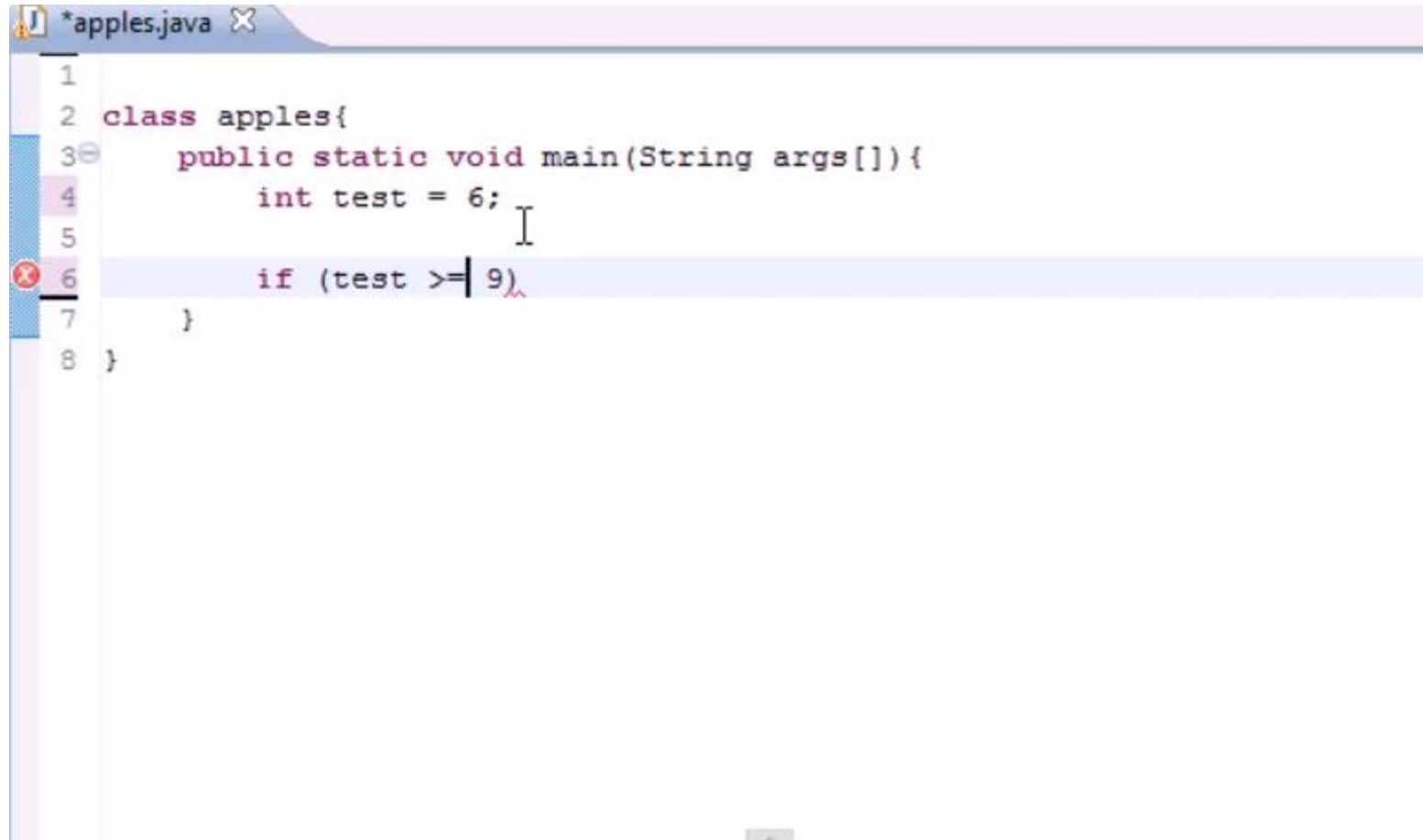
The screenshot shows a code editor window titled '\*apples.java'. The code is as follows:

```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test <= 9)
7     }
8 }
```

The code defines a class named `apples` with a `main` method. Inside the `main` method, an integer variable `test` is initialized to 6. An `if` statement is present, with the condition `test <= 9`. The code is syntactically incorrect as the `if` statement is not properly closed with a closing brace. The editor interface includes a line number margin on the left, a gutter with icons for folding and errors, and a light blue highlight on the `if` statement line.



# If-else statement in Java

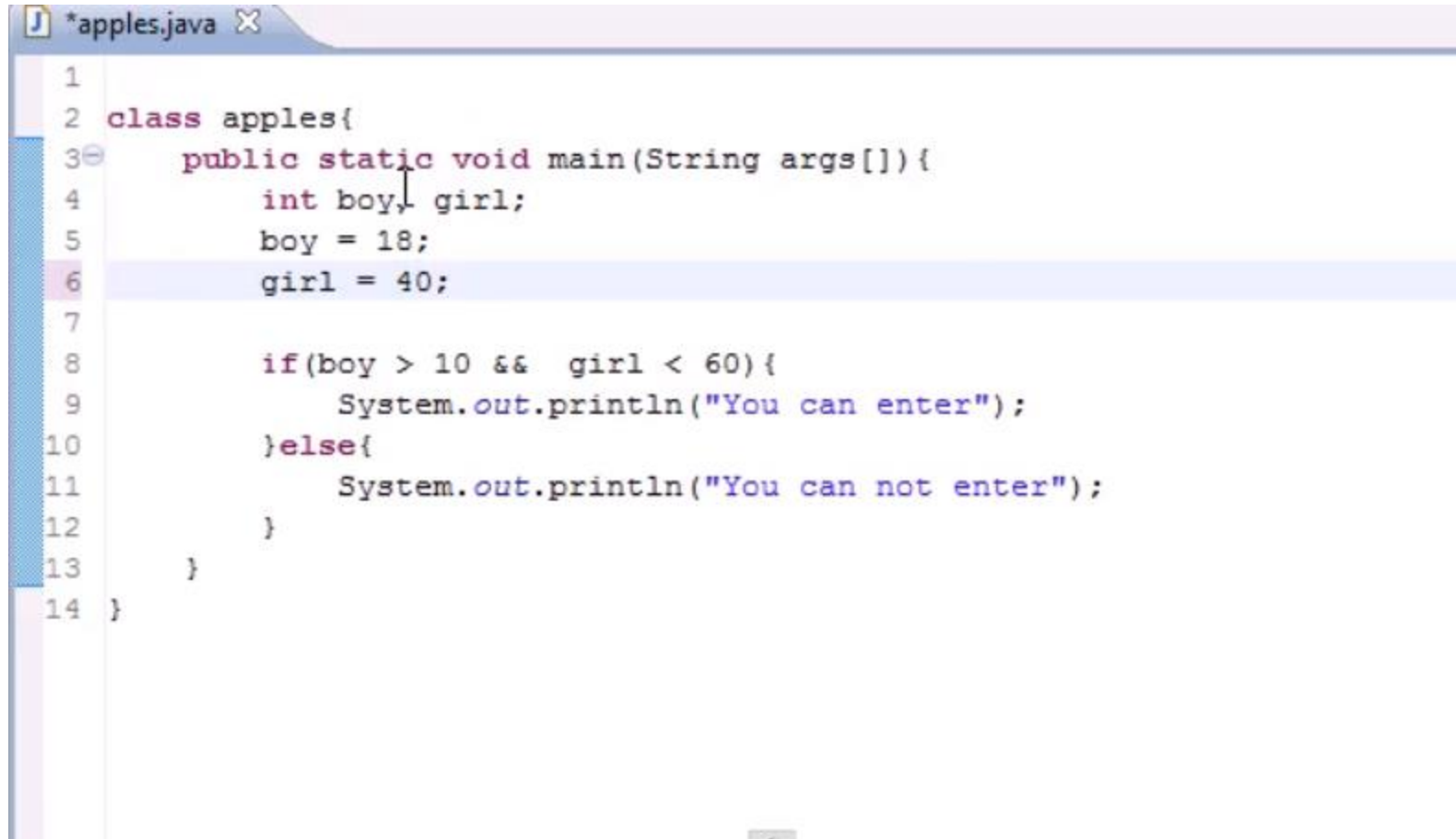


```
1
2 class apples{
3     public static void main(String args[]){
4         int test = 6;
5
6         if (test >= 9)
7     }
8 }
```

# Logical operators in Java

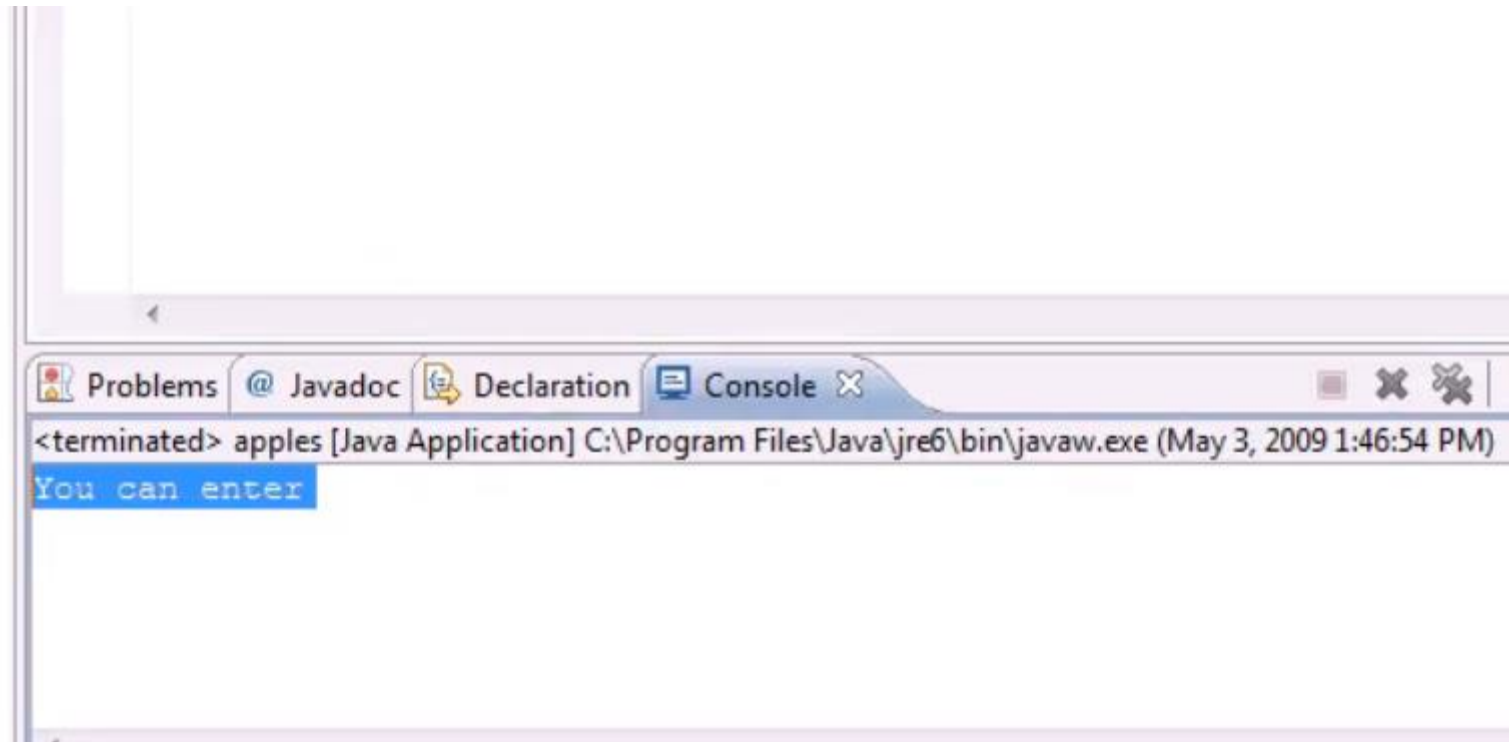
```
*apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int boy, girl;
5         boy = 18;
6         girl = 68;
7
8         if(boy > 10){
9             System.out.println("You can enter");
10        }else{
11            System.out.println("You are too young");
12        }
13    }
14 }
```

# Logical operators in Java: &&

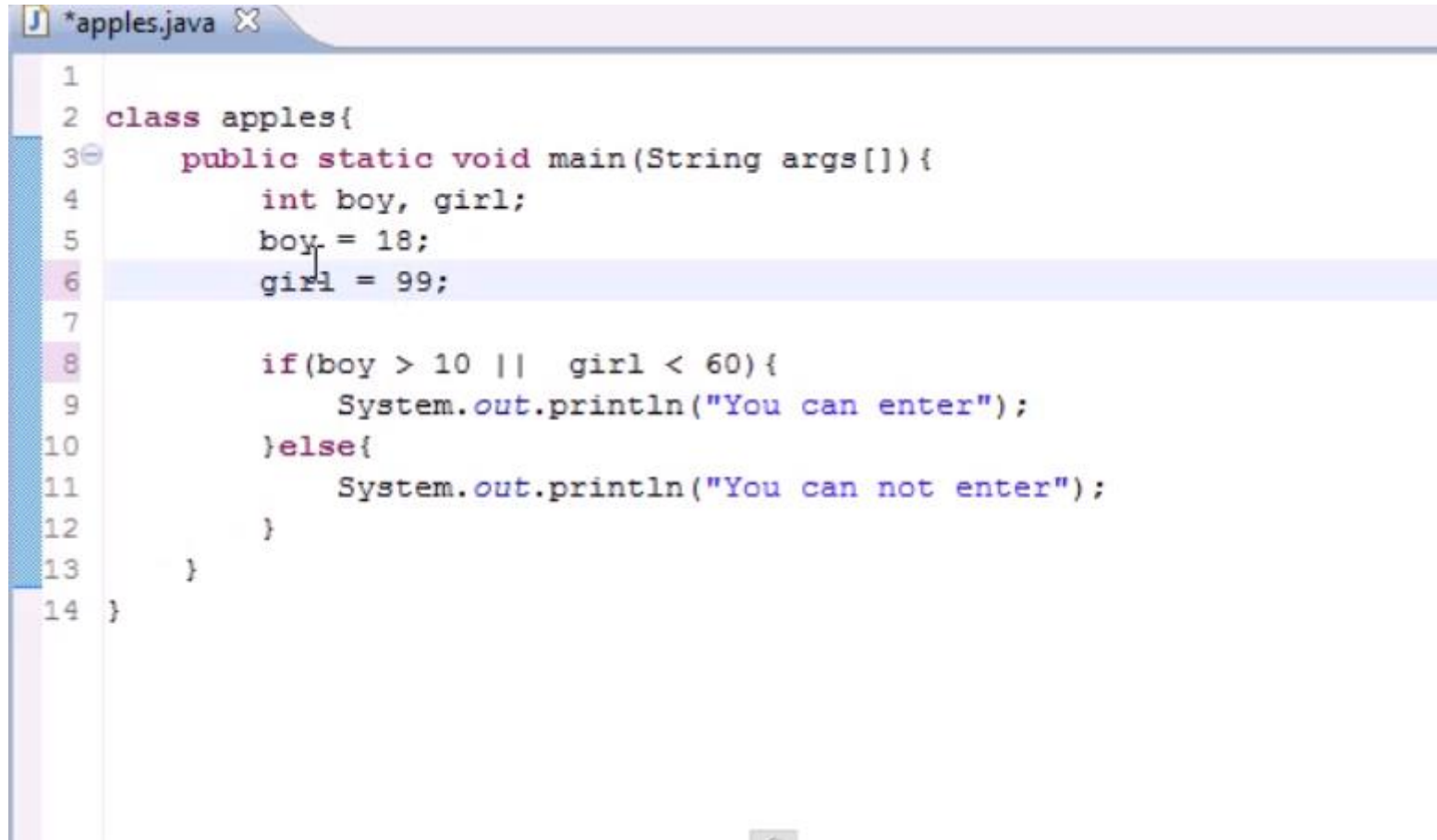


```
1
2 class apples{
3     public static void main(String args[]){
4         int boy, girl;
5         boy = 18;
6         girl = 40;
7
8         if(boy > 10 && girl < 60){
9             System.out.println("You can enter");
10        }else{
11            System.out.println("You can not enter");
12        }
13    }
14 }
```

# View output

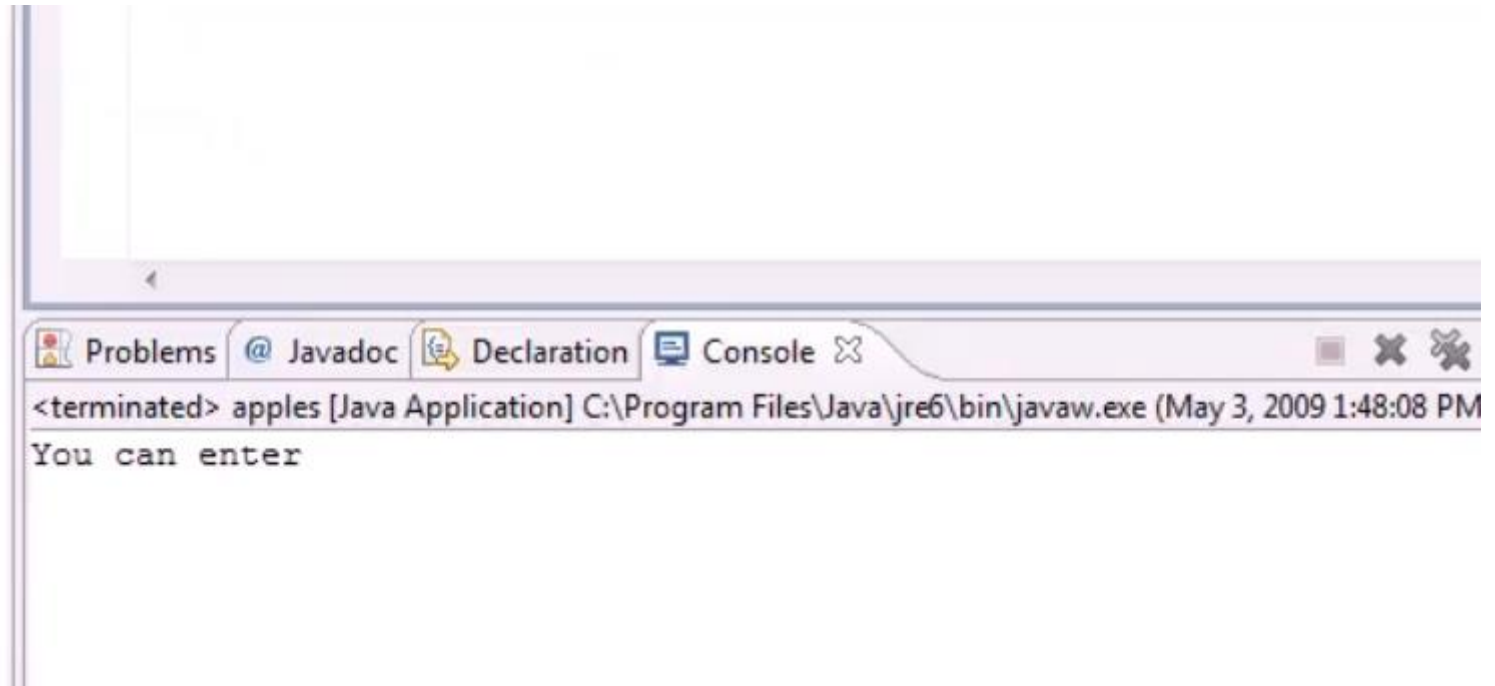


# Logical operators in Java: ||

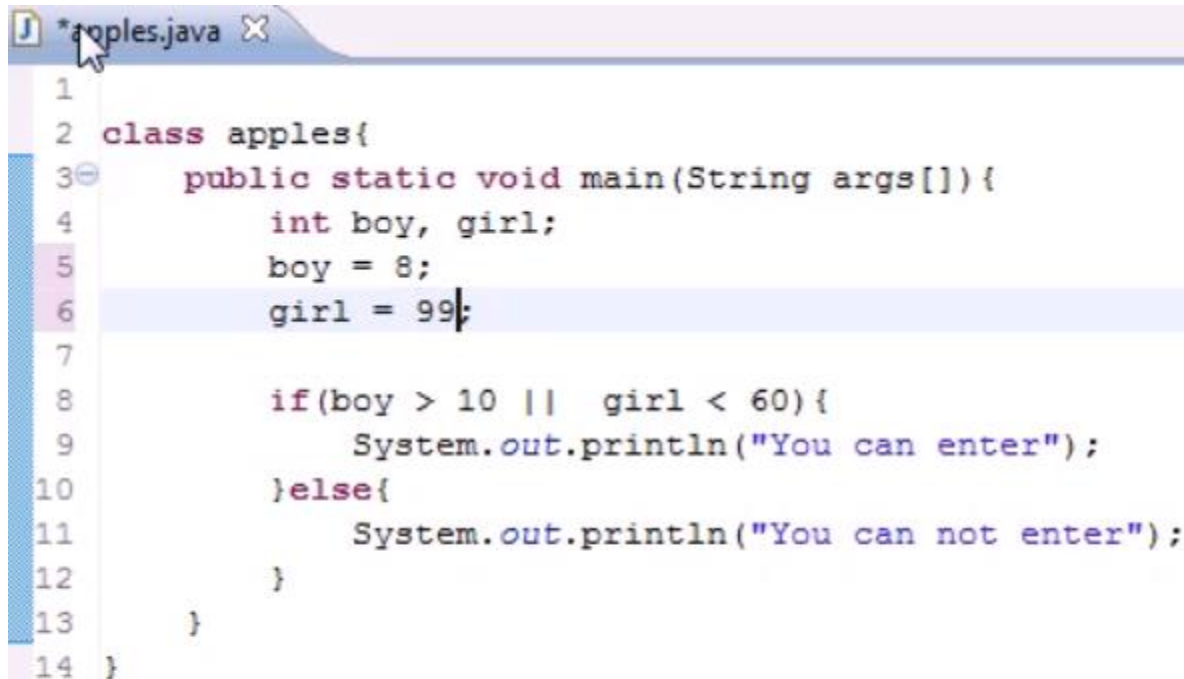


```
1
2 class apples{
3     public static void main(String args[]){
4         int boy, girl;
5         boy = 18;
6         girl = 99;
7
8         if(boy > 10 || girl < 60){
9             System.out.println("You can enter");
10        }else{
11            System.out.println("You can not enter");
12        }
13    }
14 }
```

# View output

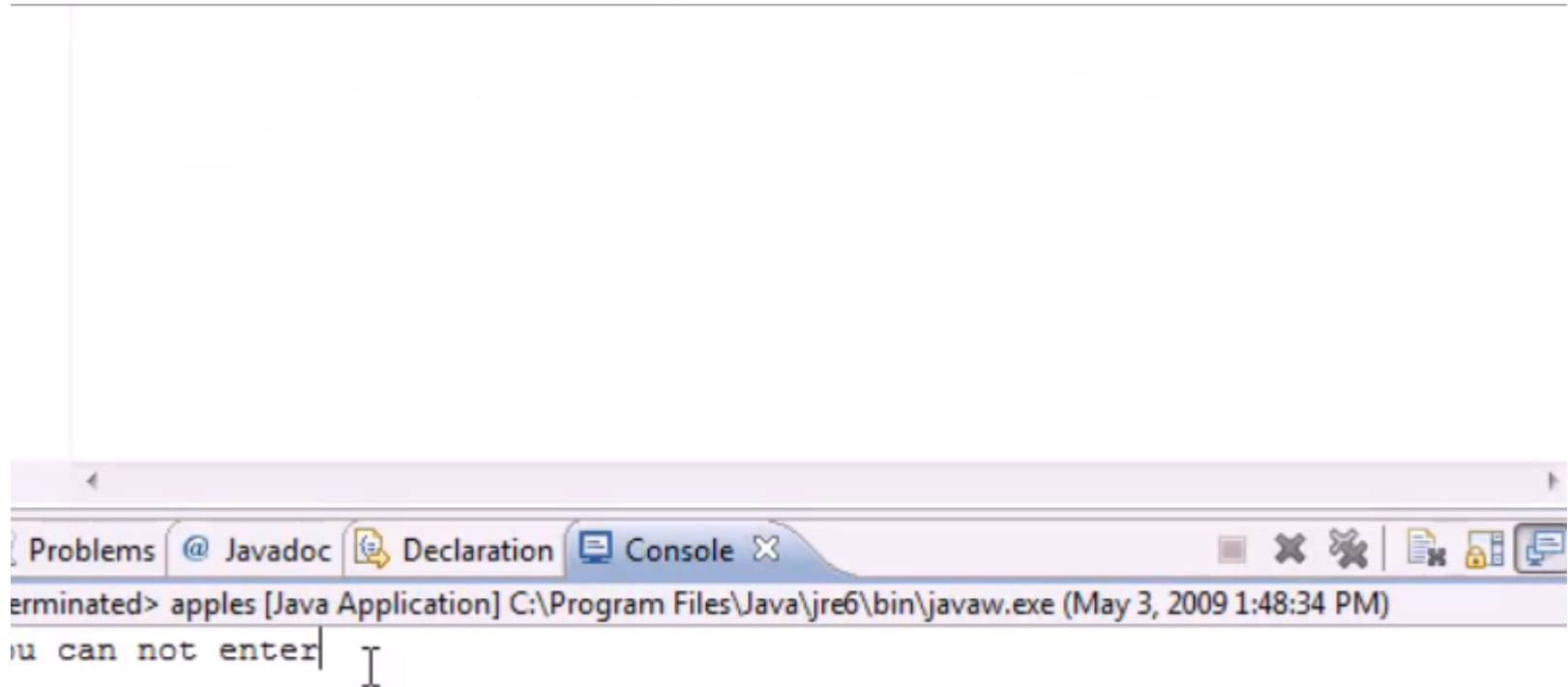


# Logical operators in Java: ||



```
1
2 class apples{
3     public static void main(String args[]){
4         int boy, girl;
5         boy = 8;
6         girl = 99;
7
8         if(boy > 10 || girl < 60){
9             System.out.println("You can enter");
10        }else{
11            System.out.println("You can not enter");
12        }
13    }
14 }
```

# View output





# Switch statement in Java

```
*apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int age;
5         age = 3;
6
7         if(age == 1){
8             System.outy.()
9         }
10        if (age ==2){}
11
12    }
13 }
```

# Switch statement in Java

```
*apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int age;
5         age = 3;
6
7         switch (age){
8             case 1:
9                 System.out.println("You can crawl");
10                break;
11             case 2:
12                 System.out.println("You can talk");
13                 break;
14             case 3:
15                 System.out.println("You can get in toruble");
16                 break;
17             default:
18
19
```

# Switch statement in Java

```
6
7      switch (age){
8      case 1:
9          System.out.println("You can crawl");
10         break;
11      case 2:
12          System.out.println("You can talk");
13         break;
14      case 3:
15          System.out.println("You can get in trouble");
16         break;
17      default:
18          System.out.println("I dont know how old you are");
19         break;
20      }
21  }
22  }
23 }
```

# View output

```
21     }  
22 }  
23 }
```

terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 3, 2009 2:09:57 PM)  
You can get in trouble

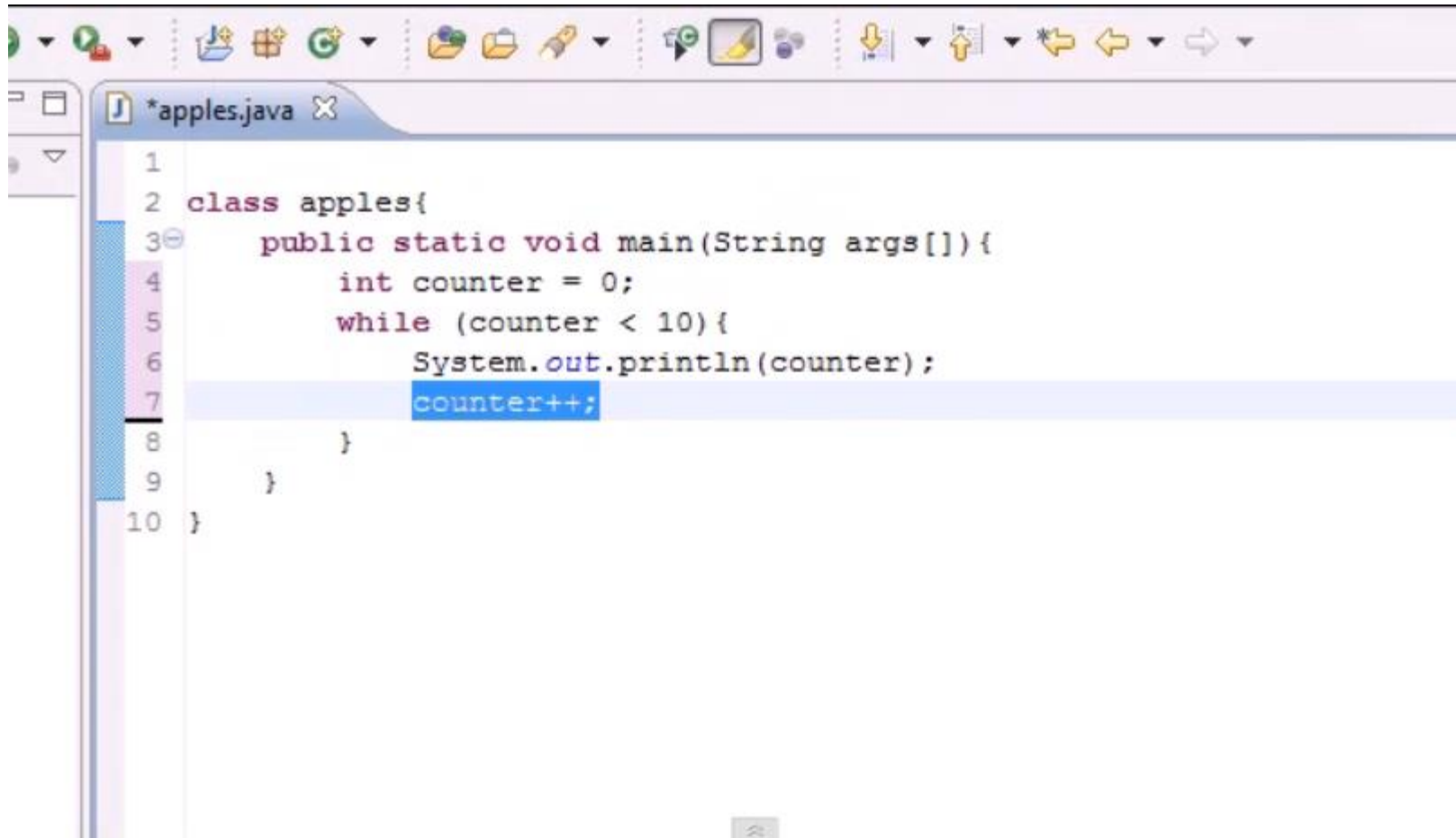
# Switch statement in Java

```
class apples{  
    public static void main(String args[]){  
        int age;  
        age = 7;  
  
        switch (age){  
        case 1:  
            System.out.println("You can crawl");  
            break;  
        case 2:  
            System.out.println("You can talk");  
            break;  
        case 3:  
            System.out.println("You can get in trouble");  
            break;  
        default:  
            System.out.println("I dont know how old you are");  
            break;  
        }  
    }  
}
```

# While loop in Java

```
apples.java X
1
2 class apples{
3     public static void main(String args[]){
4         int counter = 0;
5         while (counter < 10){
6             System.out.println(counter);
7         }
8     }
9 }
```

# While loop in Java

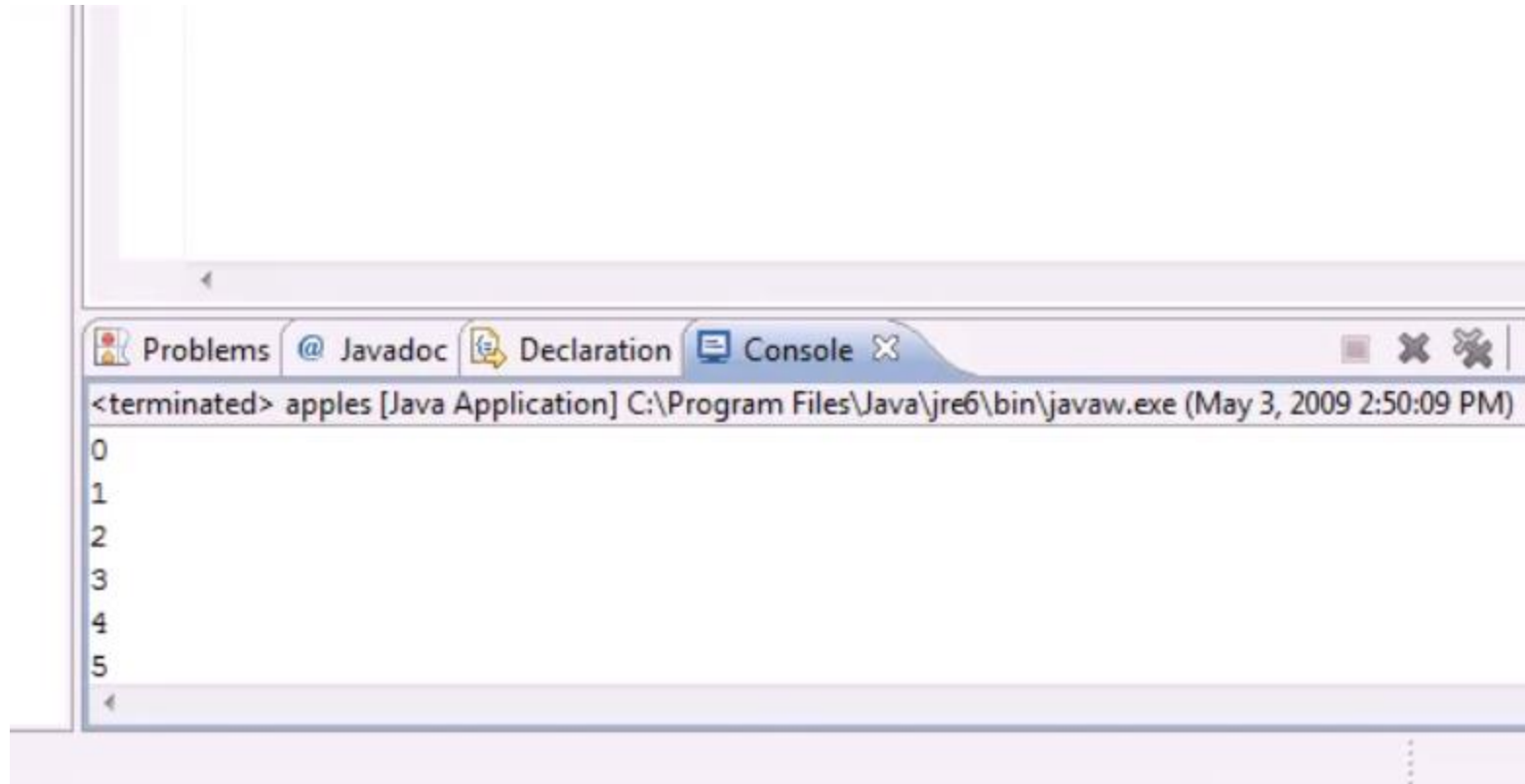


The screenshot shows a Java IDE window titled '\*apples.java'. The code is as follows:

```
1  
2 class apples{  
3     public static void main(String args[]){  
4         int counter = 0;  
5         while (counter < 10){  
6             System.out.println(counter);  
7             counter++;  
8         }  
9     }  
10 }
```

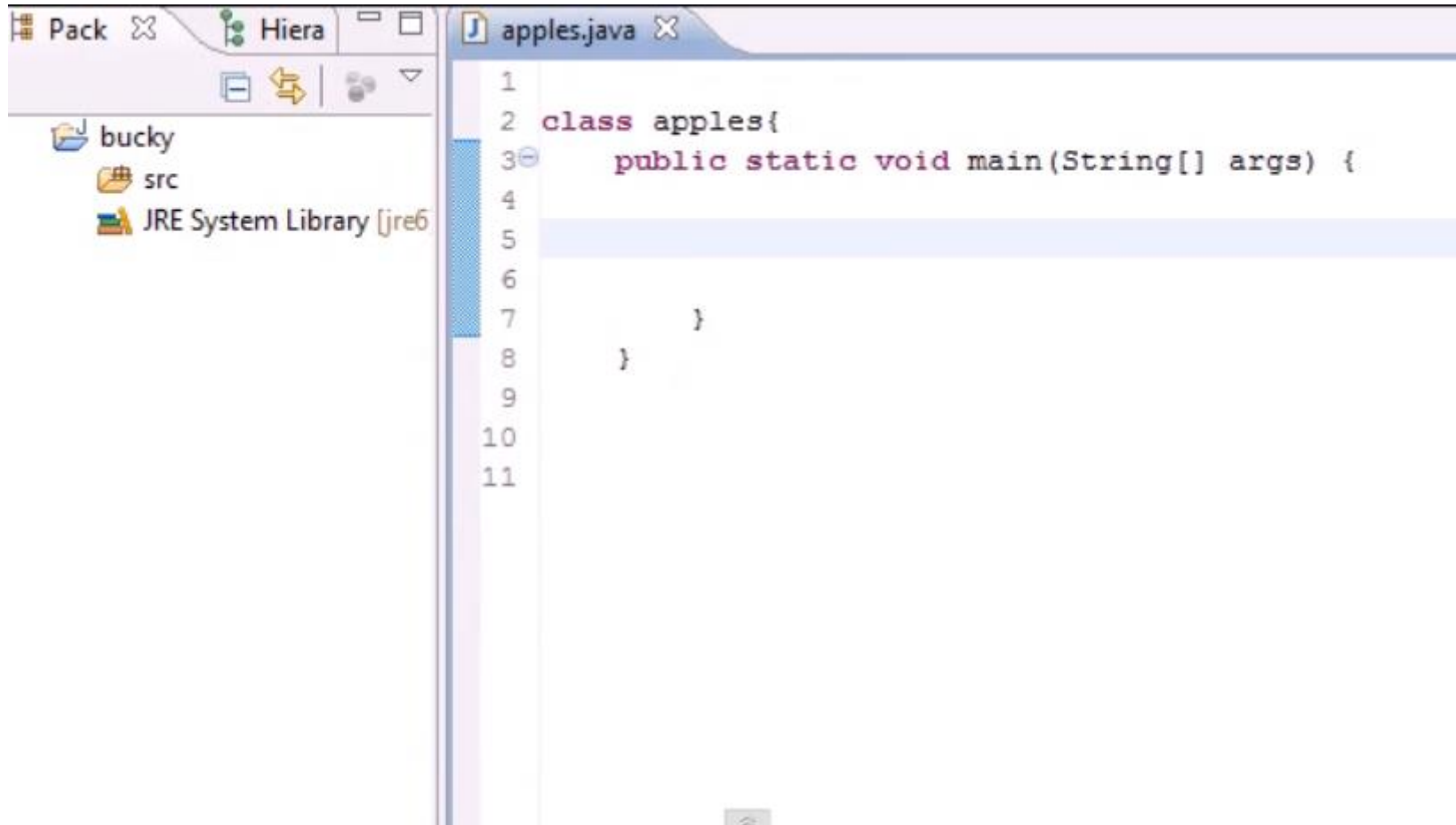
The line `counter++;` on line 7 is highlighted in blue. The IDE interface includes a toolbar at the top with various icons for file operations and a left sidebar with project explorer icons.

# View output





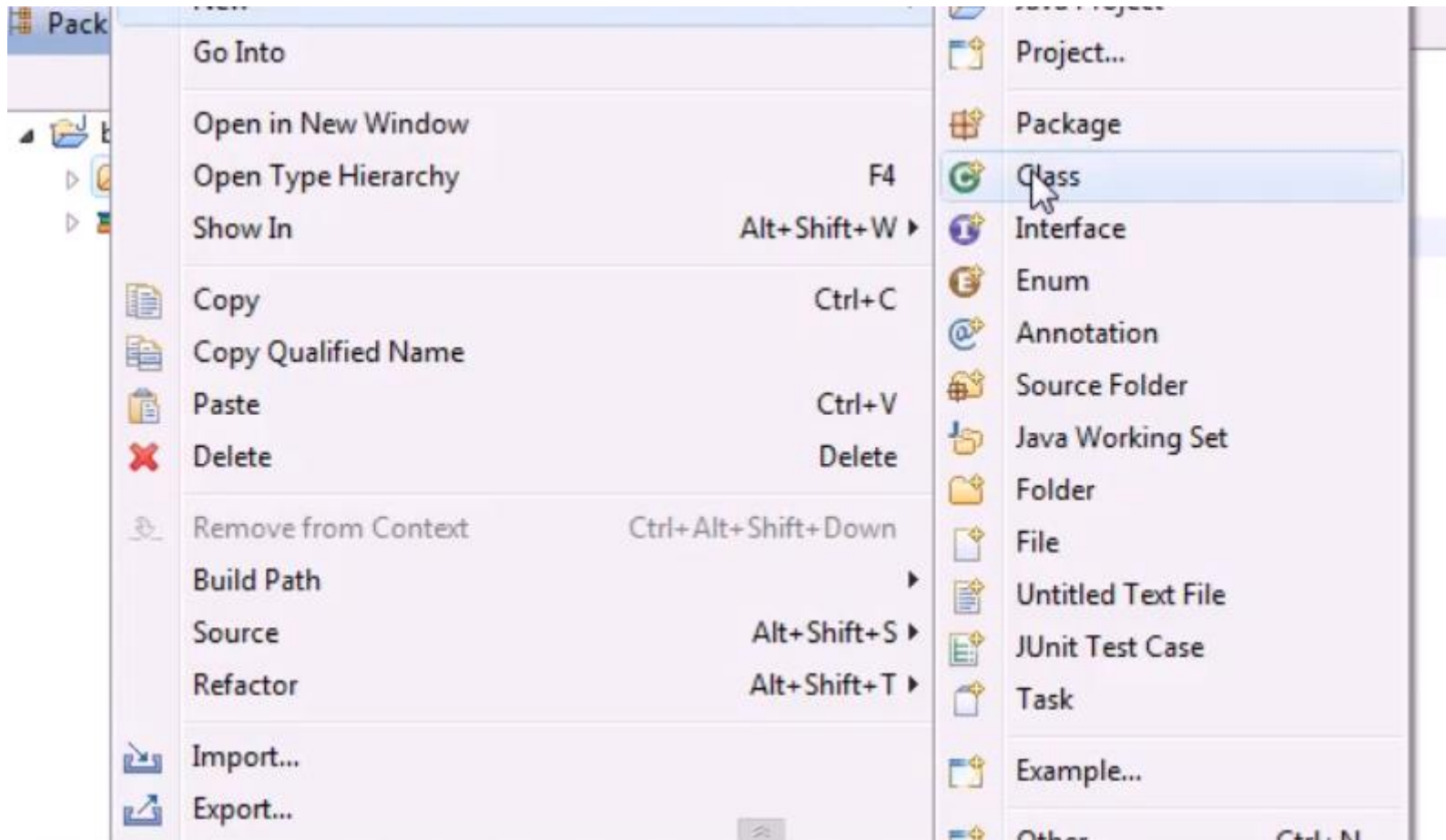
# Using objects in Java



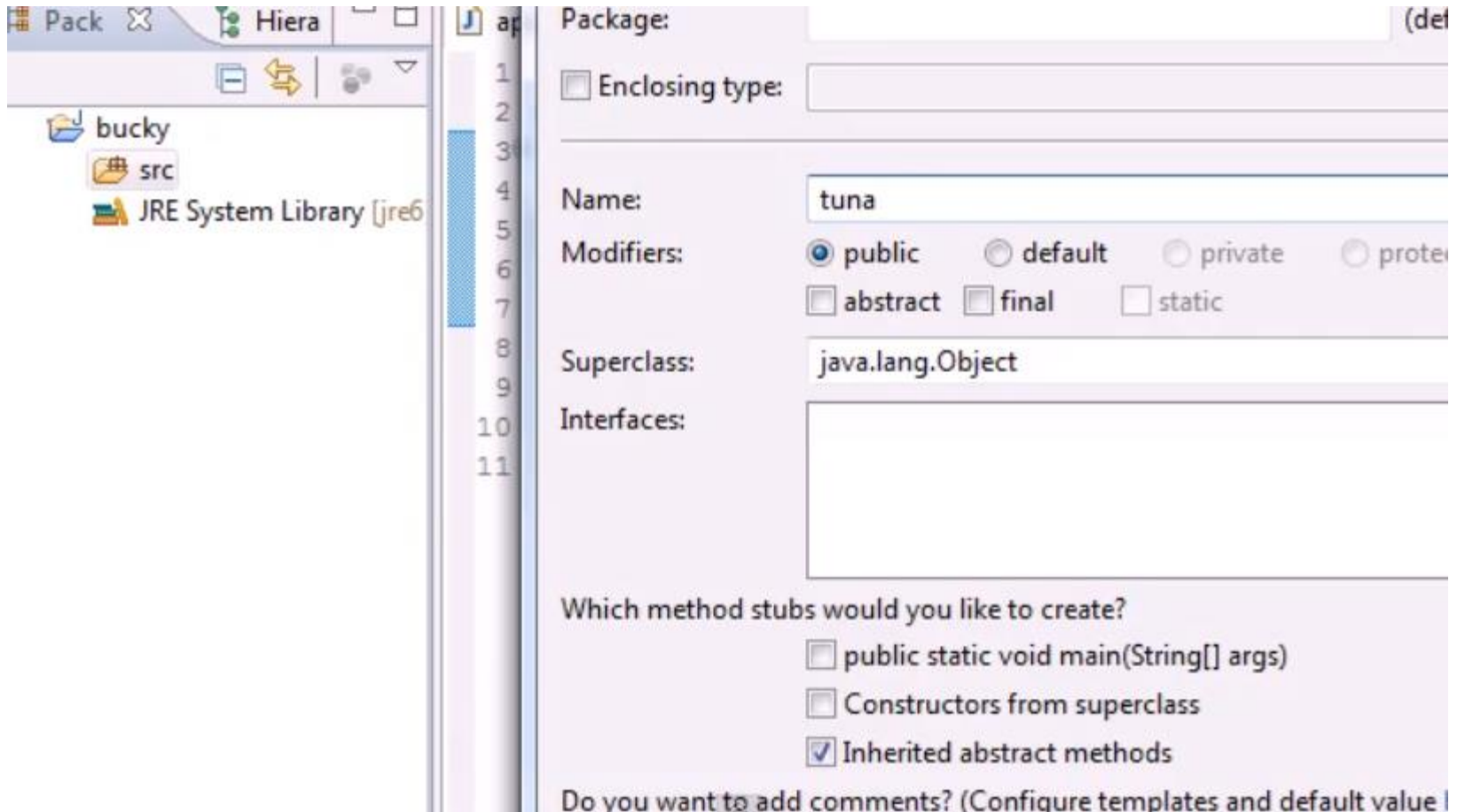
The screenshot shows an IDE window with a project named 'bucky' and a source folder 'src'. The 'JRE System Library [jre6]' is also visible. The main editor displays the file 'apples.java' with the following code:

```
1  
2 class apples{  
3     public static void main(String[] args) {  
4  
5  
6  
7     }  
8 }  
9  
10  
11
```

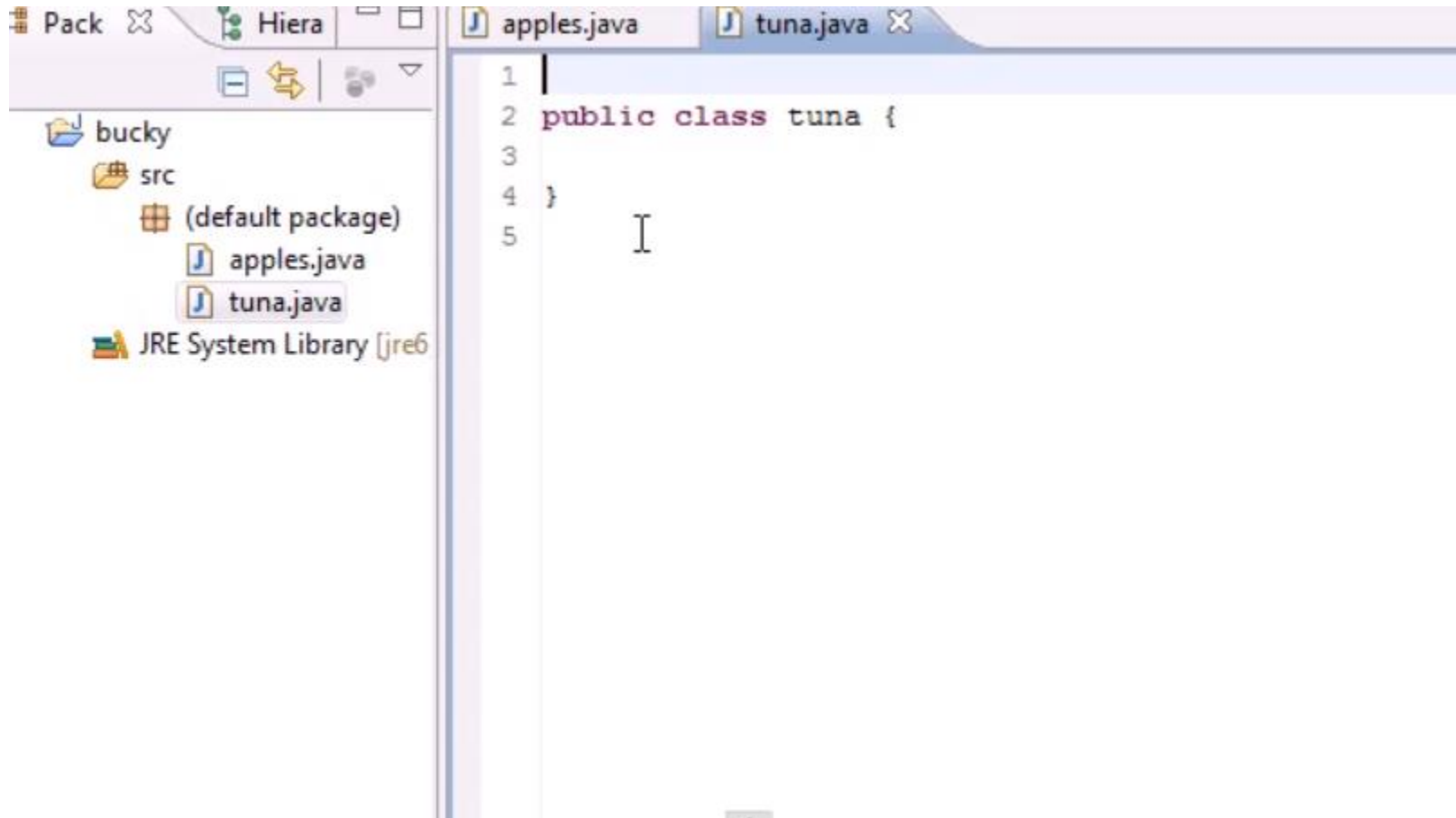
# Create a new java class



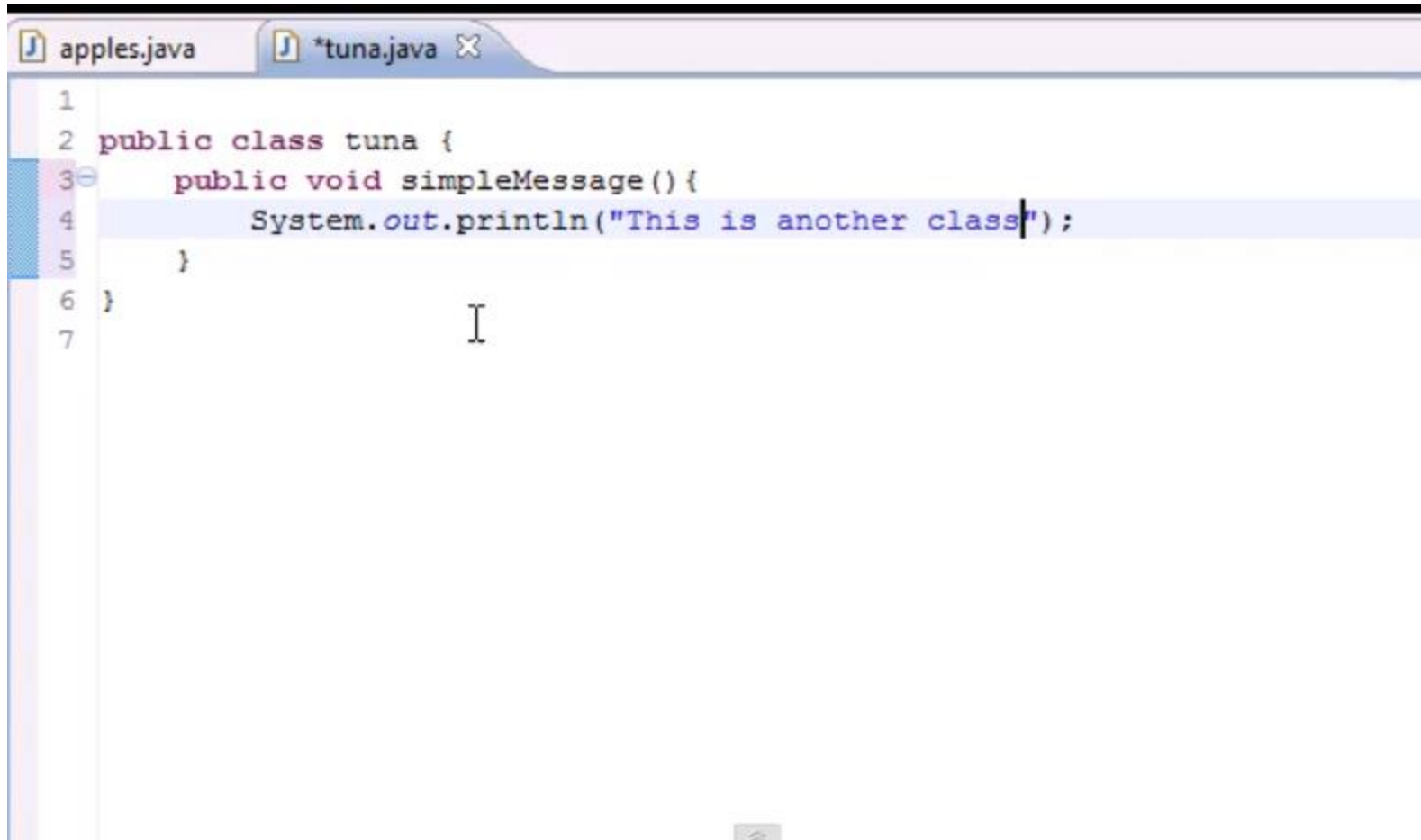
# Name the new class



# Write the java class



# Add a method in tuna

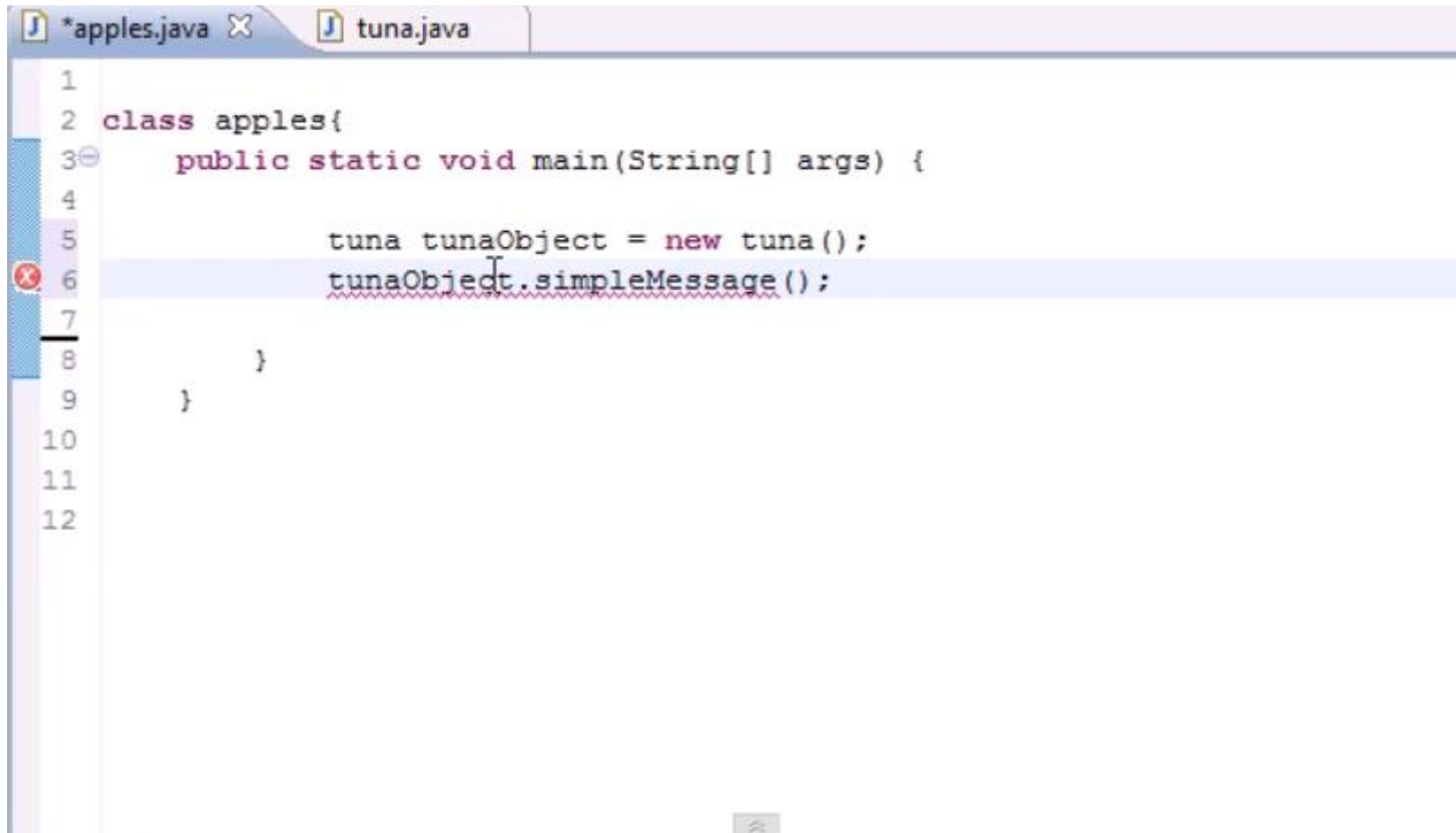


The screenshot shows a Java IDE with two tabs: 'apples.java' and '\*tuna.java'. The code in the 'tuna.java' tab is as follows:

```
1  
2 public class tuna {  
3     public void simpleMessage() {  
4         System.out.println("This is another class");  
5     }  
6 }  
7
```

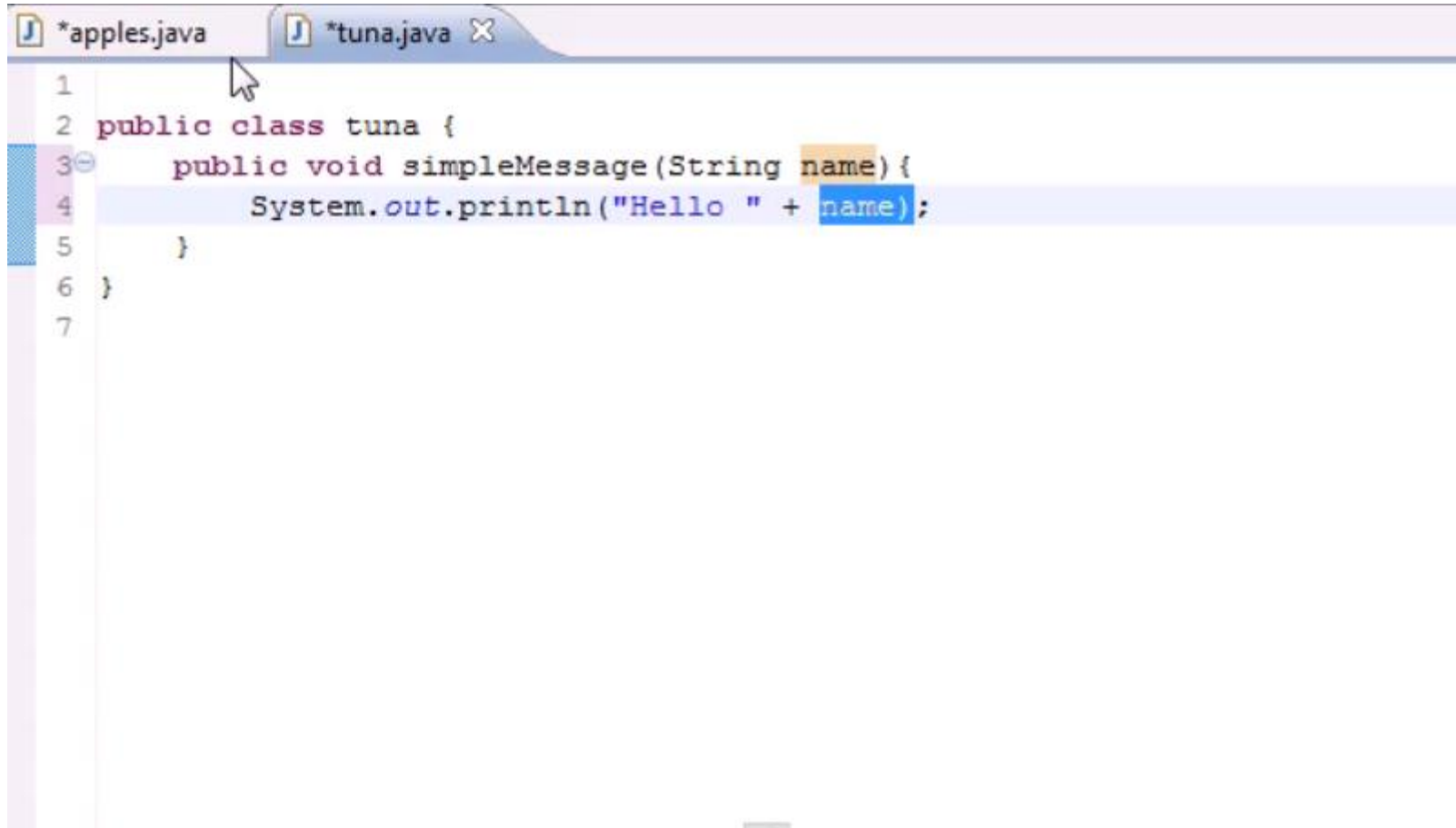
The cursor is positioned at the end of line 4, after the closing quote of the string "This is another class".

# Use objects of tuna



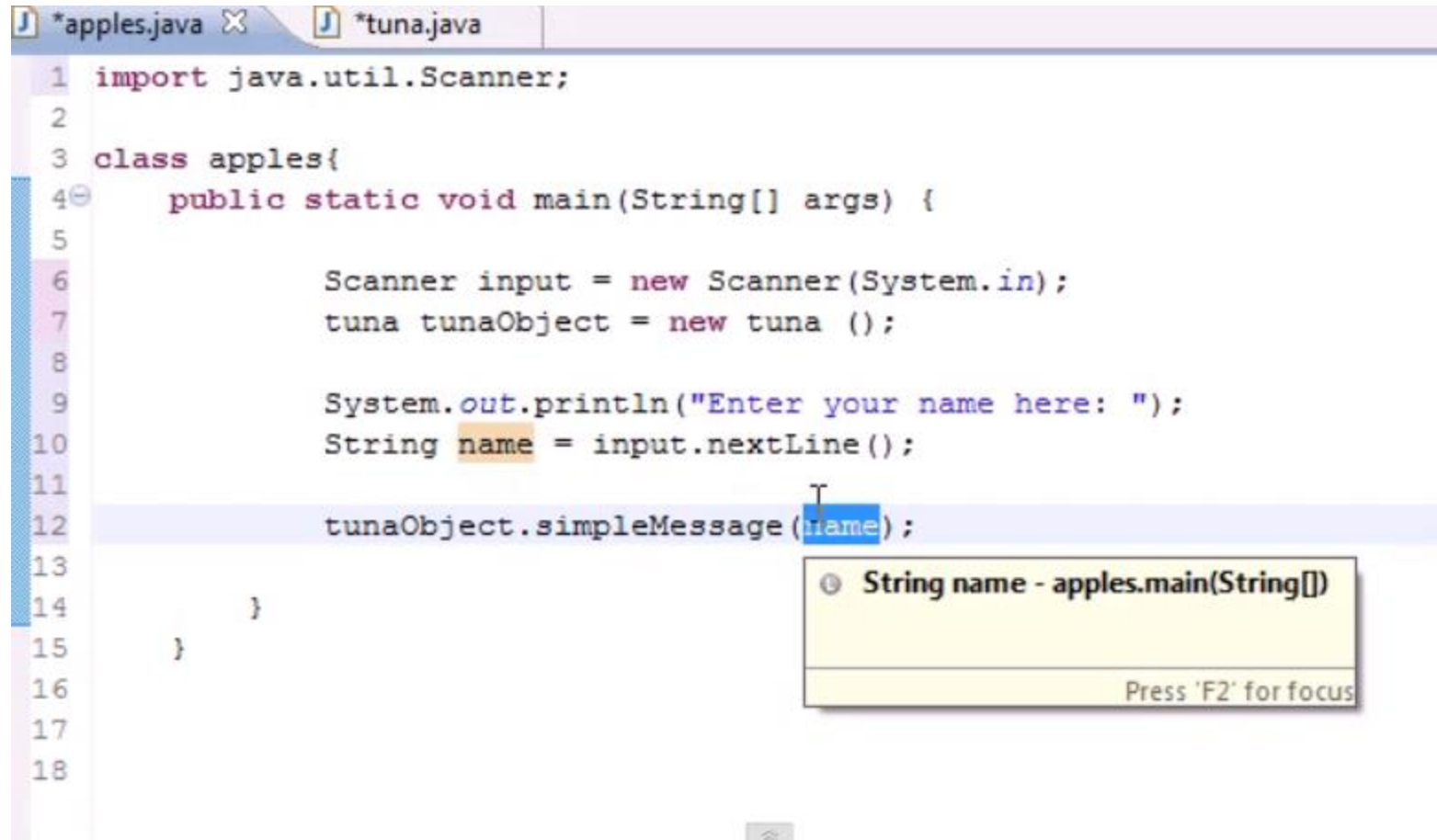
```
1
2 class apples{
3     public static void main(String[] args) {
4
5         tuna tunaObject = new tuna();
6         tunaObject.simpleMessage();
7     }
8 }
9
10
11
12
```

# Add parameter for tuna method



```
1
2 public class tuna {
3     public void simpleMessage(String name) {
4         System.out.println("Hello " + name);
5     }
6 }
7
```

# Use parameterized tuna method



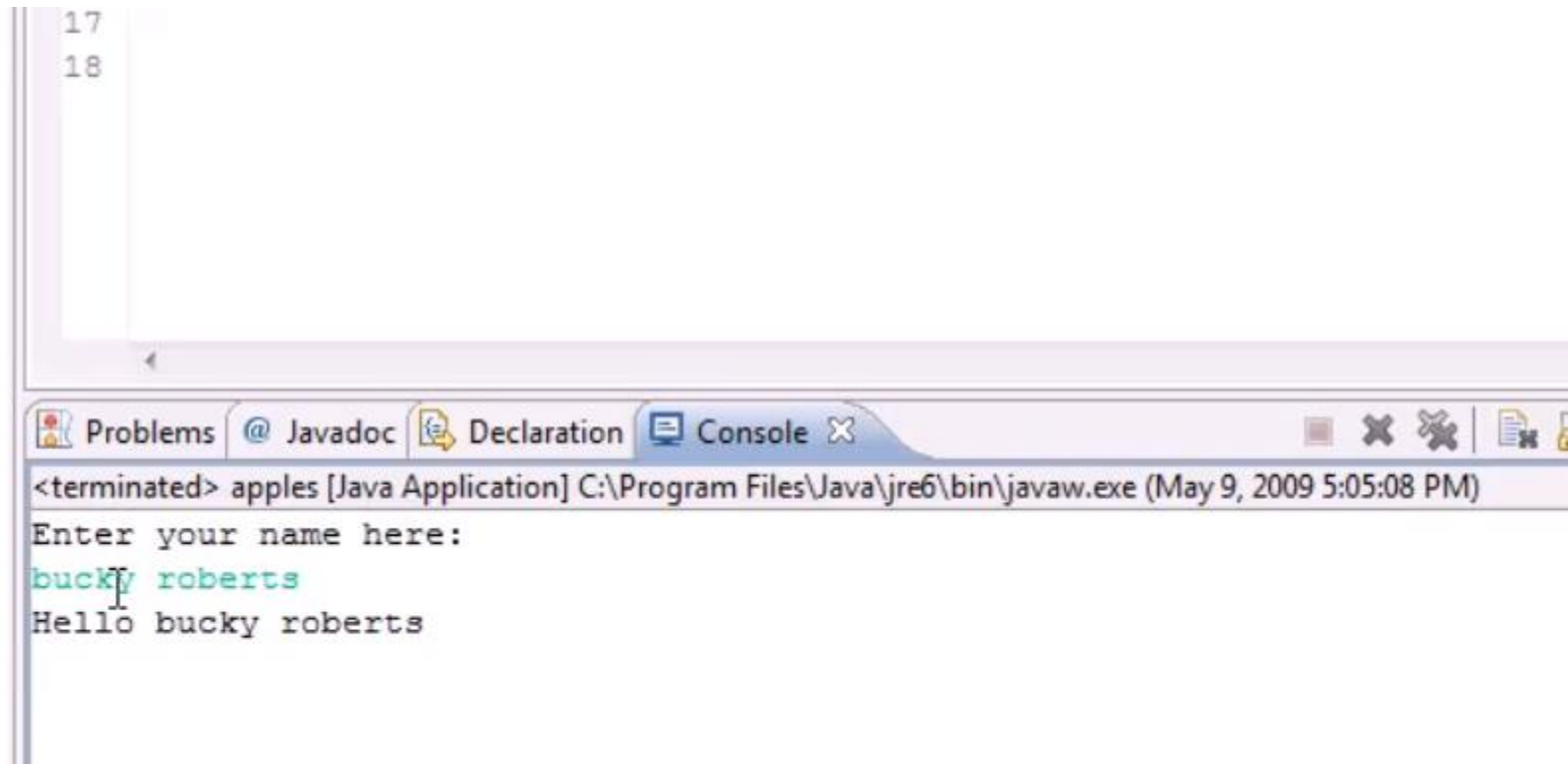
```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String[] args) {
5
6         Scanner input = new Scanner(System.in);
7         tuna tunaObject = new tuna ();
8
9         System.out.println("Enter your name here: ");
10        String name = input.nextLine();
11
12        tunaObject.simpleMessage(name);
13
14    }
15 }
16
17
18
```

String name - apples.main(String[])

Press 'F2' for focus



# View output



The screenshot shows an IDE interface with a code editor at the top and a console window at the bottom. The code editor has line numbers 17 and 18 visible. The console window has tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, showing the output of a Java application named 'apples'. The output text is: 'Enter your name here:', 'bucky roberts' (in green), and 'Hello bucky roberts'.

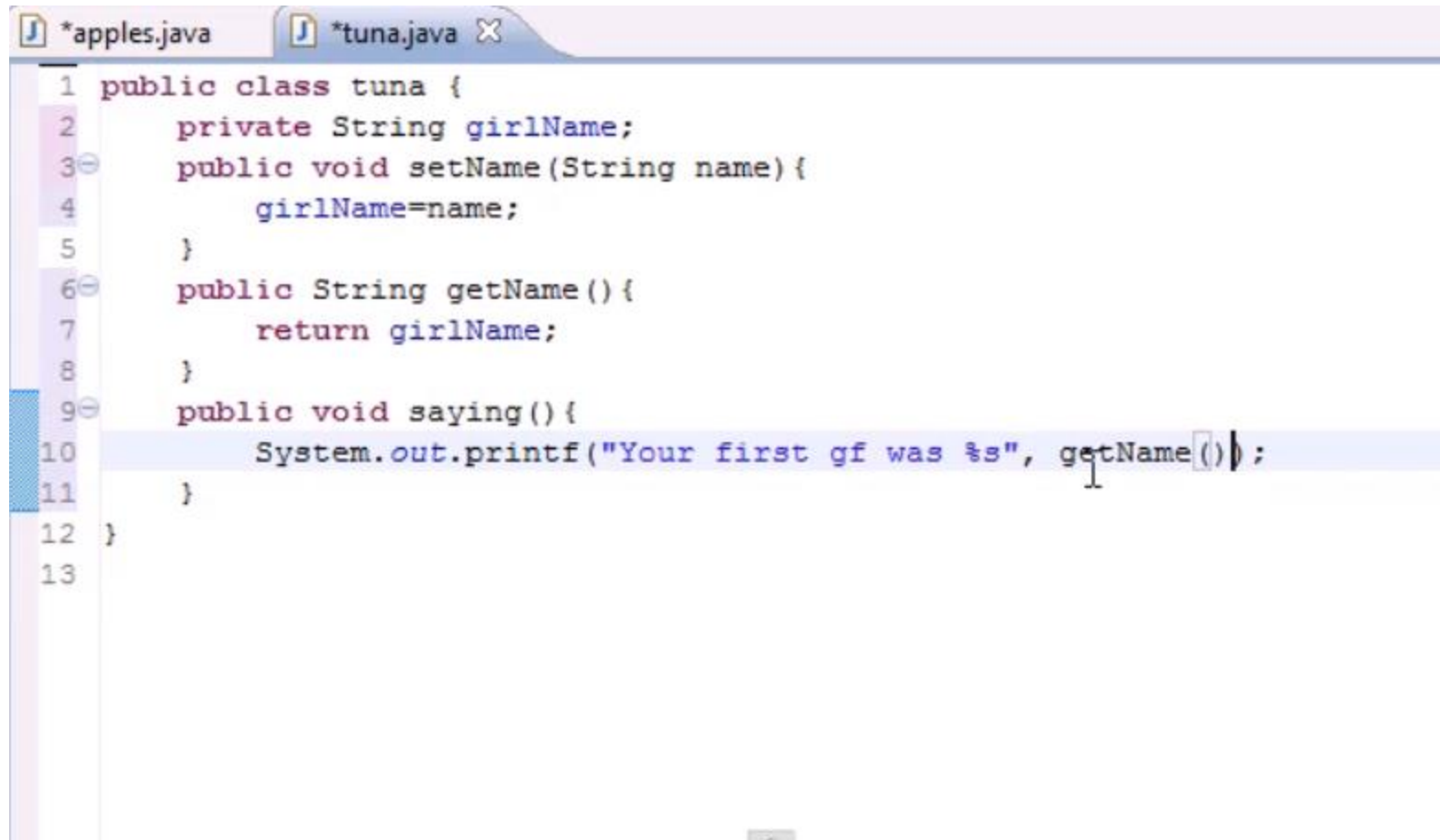
```
17  
18
```

Problems Javadoc Declaration Console X

<terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 9, 2009 5:05:08 PM)

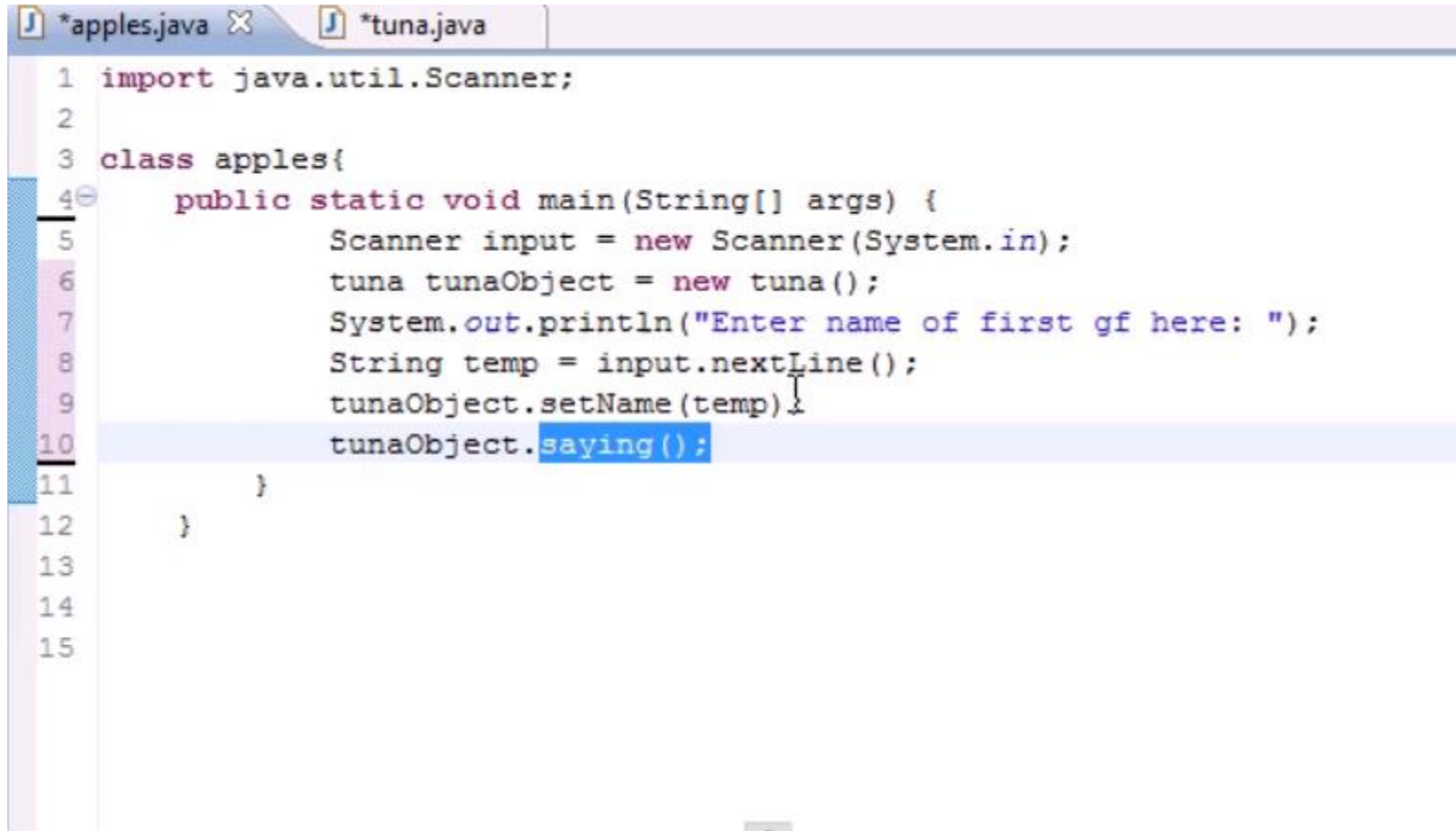
Enter your name here:  
bucky roberts  
Hello bucky roberts

# Many methods/instances in Java



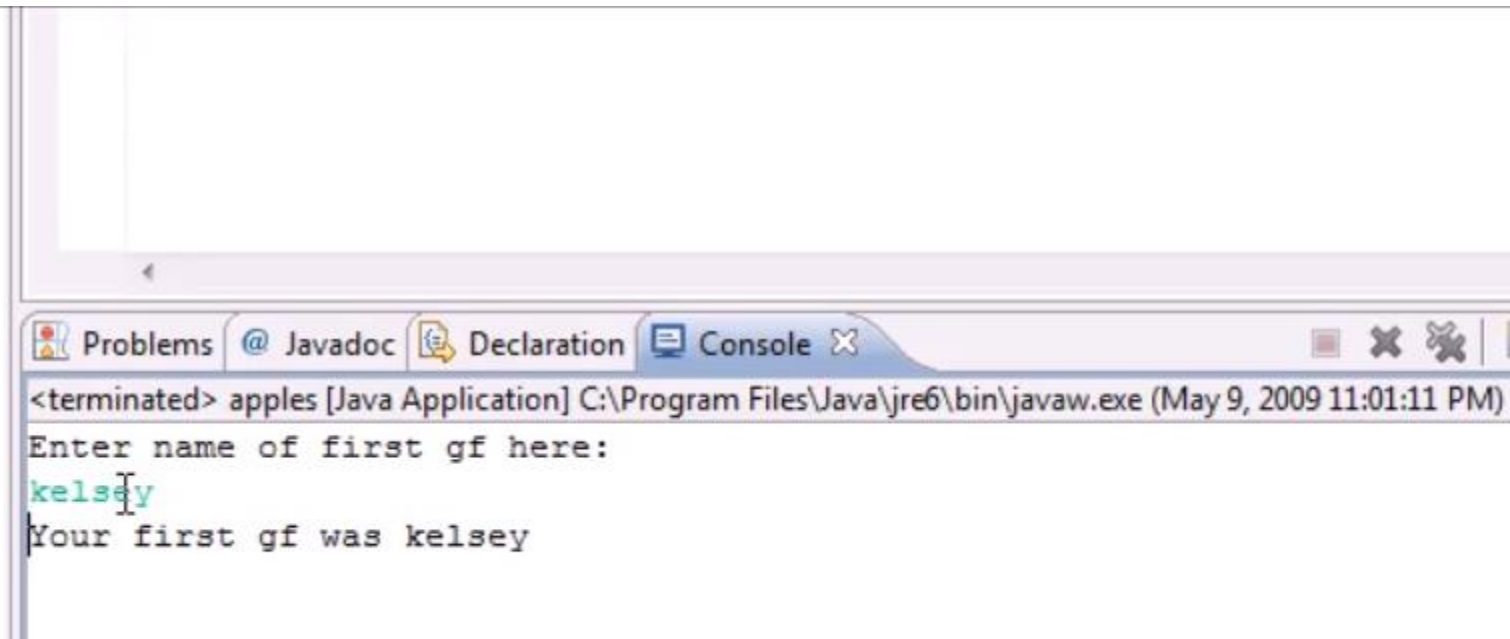
```
1 public class tuna {
2     private String girlName;
3     public void setName(String name){
4         girlName=name;
5     }
6     public String getName(){
7         return girlName;
8     }
9     public void saying(){
10        System.out.printf("Your first gf was %s", getName());
11    }
12 }
13
```

# Many methods/instances in Java



```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         tuna tunaObject = new tuna();
7         System.out.println("Enter name of first gf here: ");
8         String temp = input.nextLine();
9         tunaObject.setName(temp);
10        tunaObject.saying();
11    }
12 }
13
14
15
```

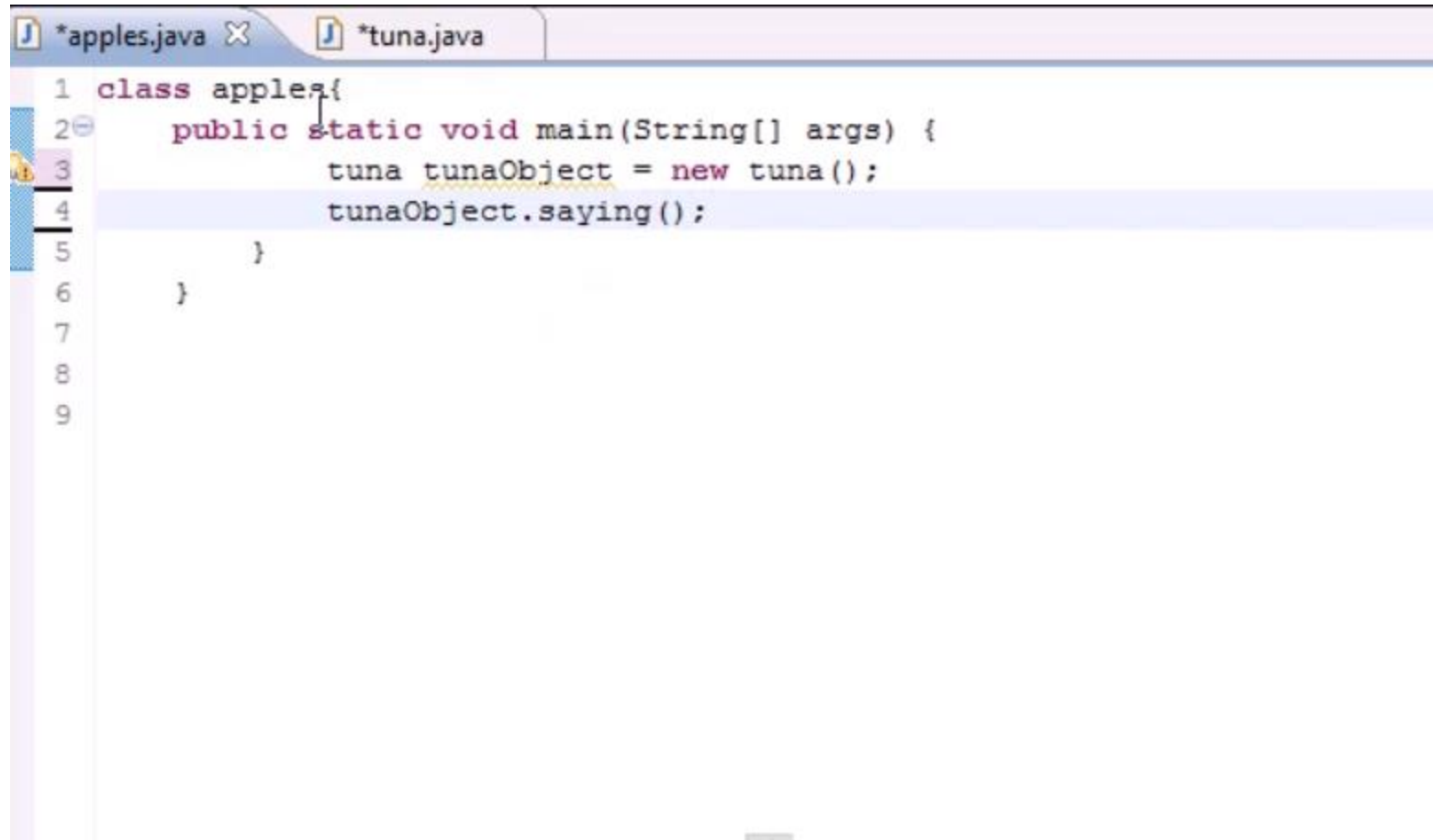
# View output



The screenshot shows an IDE window with a tab labeled "Console". The console output is as follows:

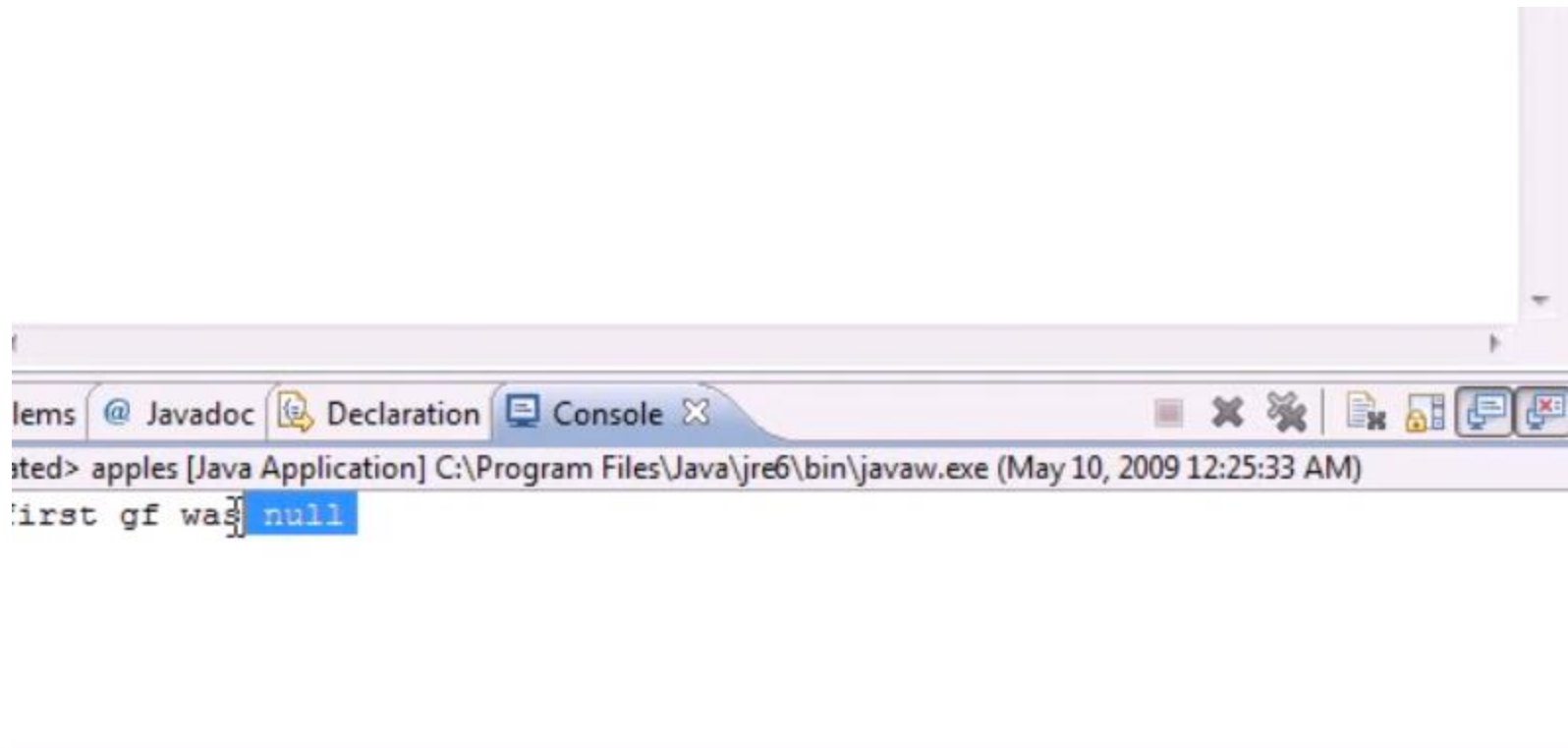
```
<terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 9, 2009 11:01:11 PM)  
Enter name of first gf here:  
kelsey  
Your first gf was kelsey
```

# Constructors in Java



```
1 class apples{
2     public static void main(String[] args) {
3         tuna tunaObject = new tuna();
4         tunaObject.saying();
5     }
6 }
7
8
9
```

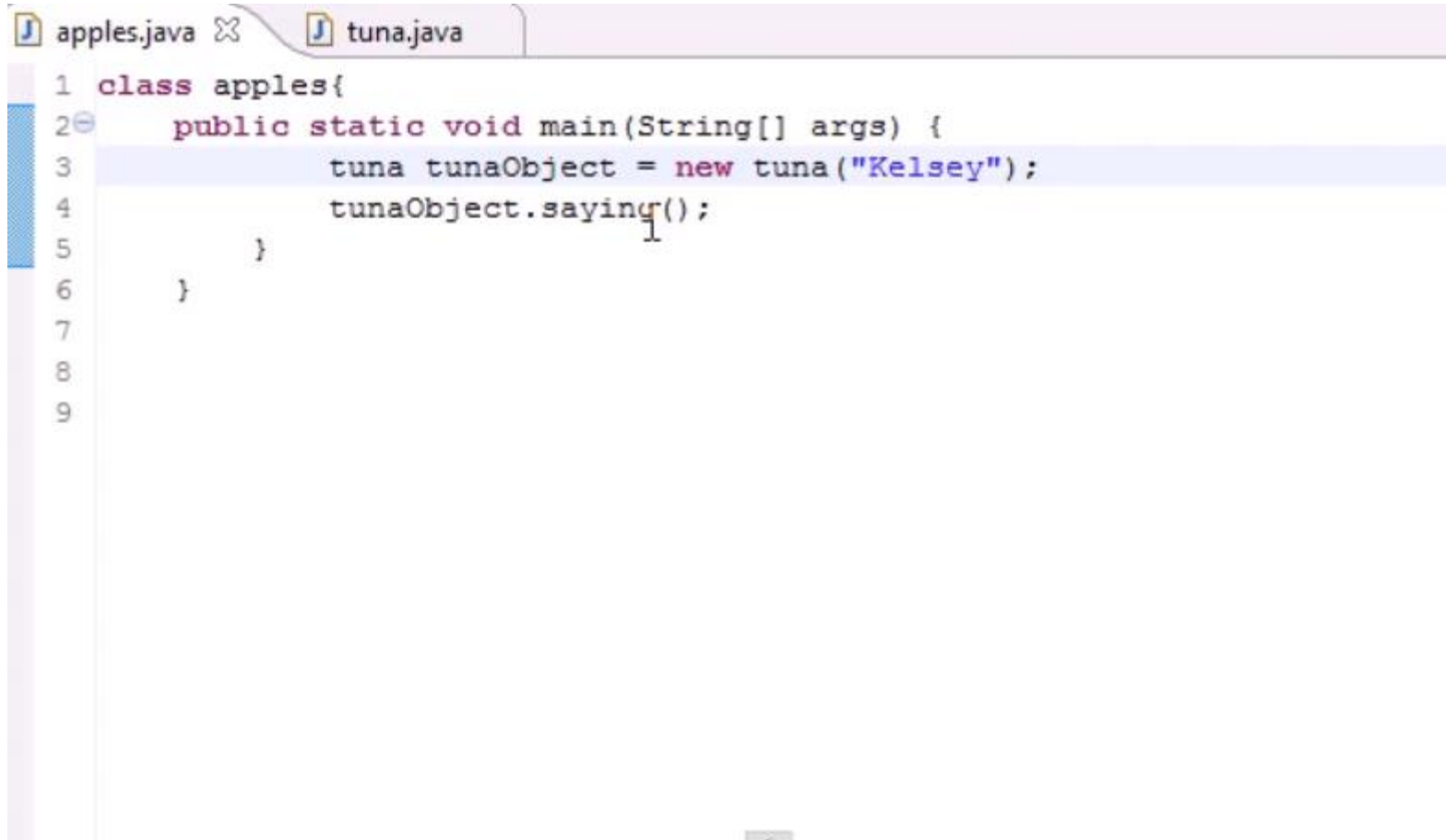
# Constructors in Java



# Add a constructors in tuna

```
apples.java *tuna.java X
1 public class tuna {
2     private String girlName;
3
4     public tuna(String name){
5         girlName=name;
6     }
7
8     public void setName(String name){
9         girlName=name;
10    }
11    public String getName(){
12        return girlName;
13    }
14    public void saying(){
15        System.out.printf("Your first gf was %s\n", getName());
16    }
17 }
18
```

# Create tuna objects using constructor



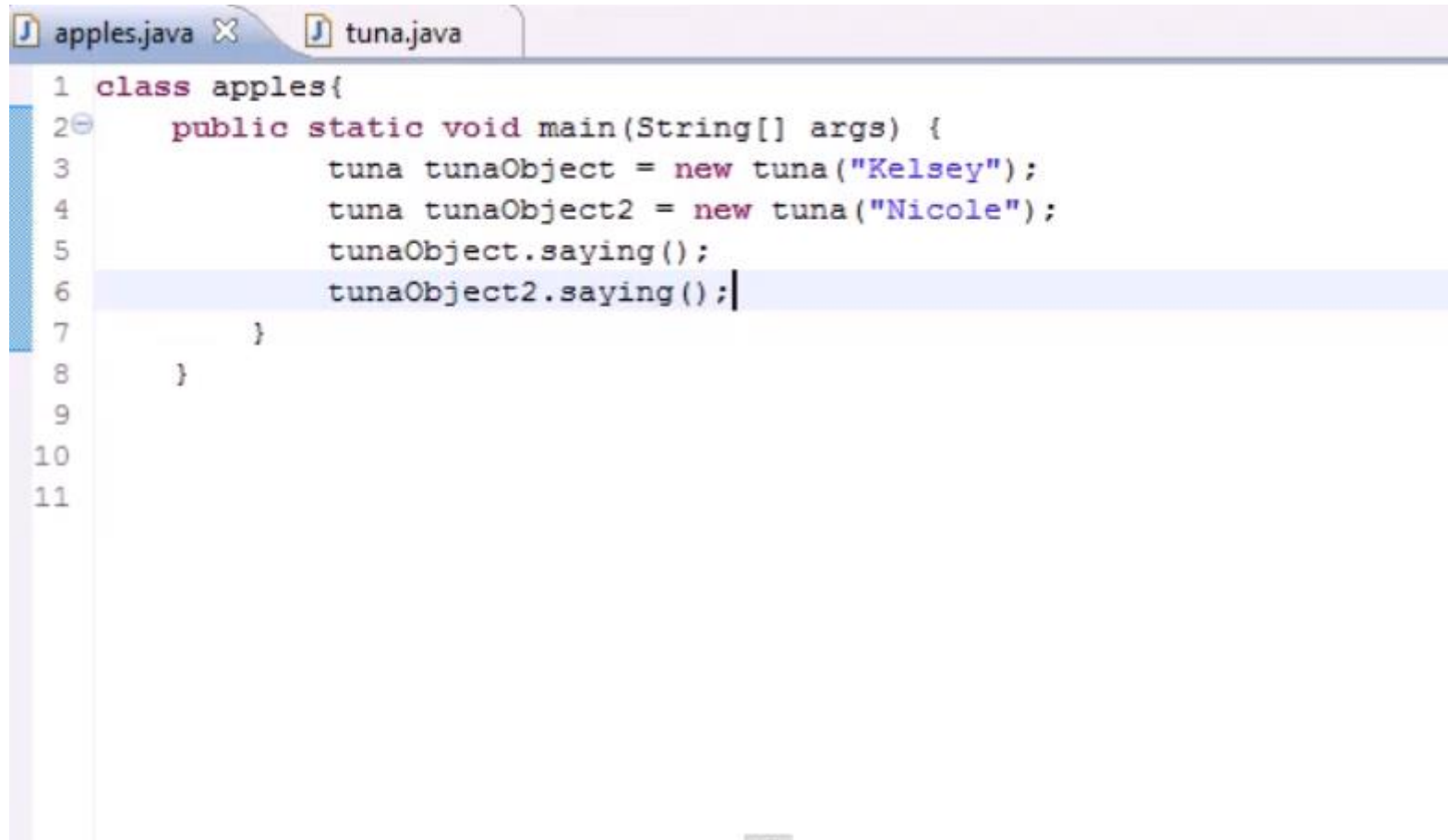
The screenshot shows a Java IDE with two tabs: `apples.java` and `tuna.java`. The `apples.java` tab is active, displaying the following code:

```
1 class apples{
2     public static void main(String[] args) {
3         tuna tunaObject = new tuna("Kelsey");
4         tunaObject.saying();
5     }
6 }
7
8
9
```

The code demonstrates the creation of a `tuna` object named `tunaObject` using the `new` keyword and the `tuna` constructor, passing the string `"Kelsey"` as an argument. The `tunaObject.saying();` line indicates a method call on the created object.

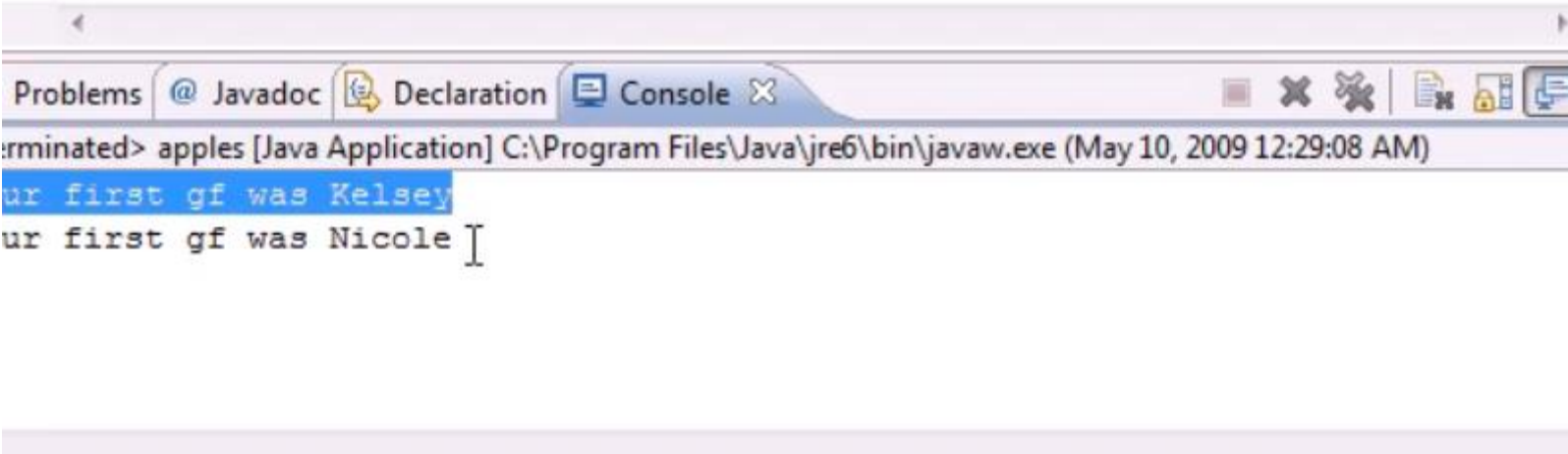


# Call tuna methods



```
1 class apples{
2     public static void main(String[] args) {
3         tuna tunaObject = new tuna("Kelseey");
4         tuna tunaObject2 = new tuna("Nicole");
5         tunaObject.saying();
6         tunaObject2.saying();
7     }
8 }
9
10
11
```

# Constructors in Java



The screenshot shows a Java IDE's console window. The title bar includes tabs for 'Problems', '@ Javadoc', 'Declaration', and 'Console'. The console text shows the program has terminated and displays two lines of output: 'ur first gf was Kelsey' (highlighted in blue) and 'ur first gf was Nicole' (with a cursor at the end).

```
terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 10, 2009 12:29:08 AM)  
ur first gf was Kelsey  
ur first gf was Nicole I
```

# What Is a Class?

- A *class* is the blueprint from which individual objects are created.
- Classes encapsulates states and behaviors of single entity
- Class includes variables which are called instance variables
- Class have methods which are called instance methods

# Declaring Classes

Classes are defined in the following way:

```
class MyClass {  
    // field, constructor, and  
    // method declarations  
}
```

# Declaring Classes

Class definition consists of two parts:

- The *class declaration*.
- The *class body* (the area between the braces) contains **all** the code that provides for the life cycle of the objects created from the class
- Class body can have **constructors for initializing new objects, declarations for the fields that provide the state of the class, and methods to implement the behavior of the class.**

# Declaring Classes

Class definition consists of two parts:

- The *class declaration*.
- The *class body* (the area between the braces) contains **all** the code that provides for the life cycle of the objects created from the class
- Class body can have **constructors for initializing new objects, declarations for the fields that provide the state of the class, and methods to implement the behavior of the class.**

# Declaring Classes

Class declaration can include these components-

- Modifiers such as *public*, *private*
- The class name, with the initial letter capitalized by convention.
- The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only *extend* (subclass) one parent.
- Followed by The class body, surrounded by braces, {}.

# Declaring Classes

We can also add modifiers like *public* or *private*

- The modifiers *public* and *private*, which determine what other classes can access the declared class
- We **usually write public** which means any other class can access



# What is an object

- An object stores its state in *fields* (instance variables)
- An object exposes its behavior through *methods*
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.

# What is an object

- Hiding internal state and requiring **all** interaction to be performed through an object's methods is known as *data encapsulation* — a fundamental principle of object-oriented programming

# Declaring Member Variables

There are several kinds of variables:

- Member variables in a class—these are called *fields*.
- Variables in a method or block of code—these are called *local variables*.
- Variables in method declarations—these are called *parameters*.

# Declaring Member Variables

Field declarations are composed of three components, in order:

- Zero or more modifiers, such as public or private.
- The field's type.
- The field's name.

# Access Modifiers

The first (left-most) modifier used lets you control what other classes have access to a member field.

- public modifier—the field is accessible from all classes.
- private modifier—the field is accessible only within its own class.
- In the spirit of encapsulation, it is common to make fields private.

# Variable Types

All variables must have a type.

- You can use primitive types such as int, float, boolean, etc. Or
- you can use reference types, such as strings, arrays, or objects.

# Variable Names

All variables must have a type.

- You can use primitive types such as int, float, boolean, etc. Or
- you can use reference types, such as strings, arrays, or objects.

# Defining Methods

Method declarations have six components, in order:

- **Modifiers**—such as public, private, and others you will learn about later.
- **The return type**—the data type of the value returned by the method, or void if the method does not return a value.
- **The method name**—the rules for field names apply to method names as well, but the convention is a little different.



# Defining Methods

Method declarations have six components, in order:

- The **method body, enclosed between braces**—the method's code, including the declaration of local variables, goes here.

# Nested if in Java

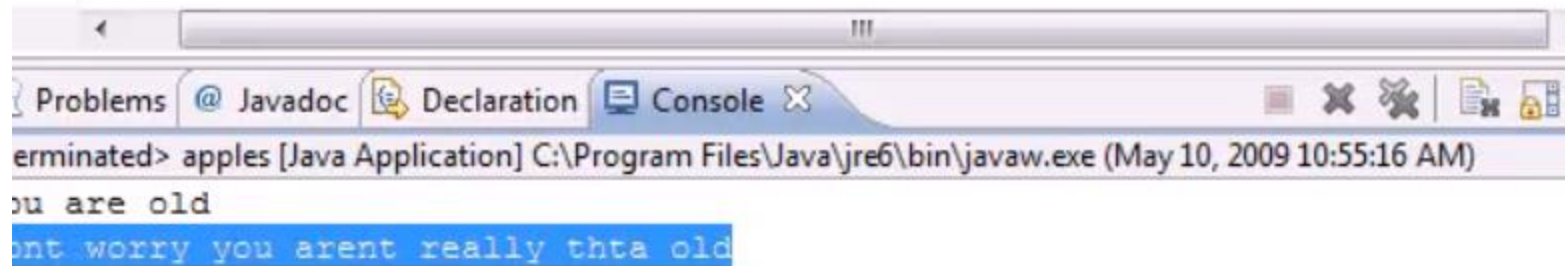
```
*apples.java X tuna.java
1 class apples{
2     public static void main(String[] args) {
3         int age = 60;
4
5         if (age < 50){
6             System.out.println("You are young");
7         }else{
8             System.out.println("You are old");
9             if (age > 75){
10                System.out.println("You are REALLY old!");
11            }
12        }
13    }
14 }
15
16
17
```

# Nested if in Java

```
*apples.java X  tuna.java
1  ss apples{
2  public static void main(String[] args) {
3      int age = 60;
4
5      if (age < 50){
6          System.out.println("You are young");
7      }else{
8          System.out.println("You are old");
9          if (age > 75){
10             System.out.println("You are REALLY old!");
11         }else{
12             System.out.println("dont worry you arent really thta old");
13         }
14     }
15 }
16 }
17
18
19
```

# View output

18  
19

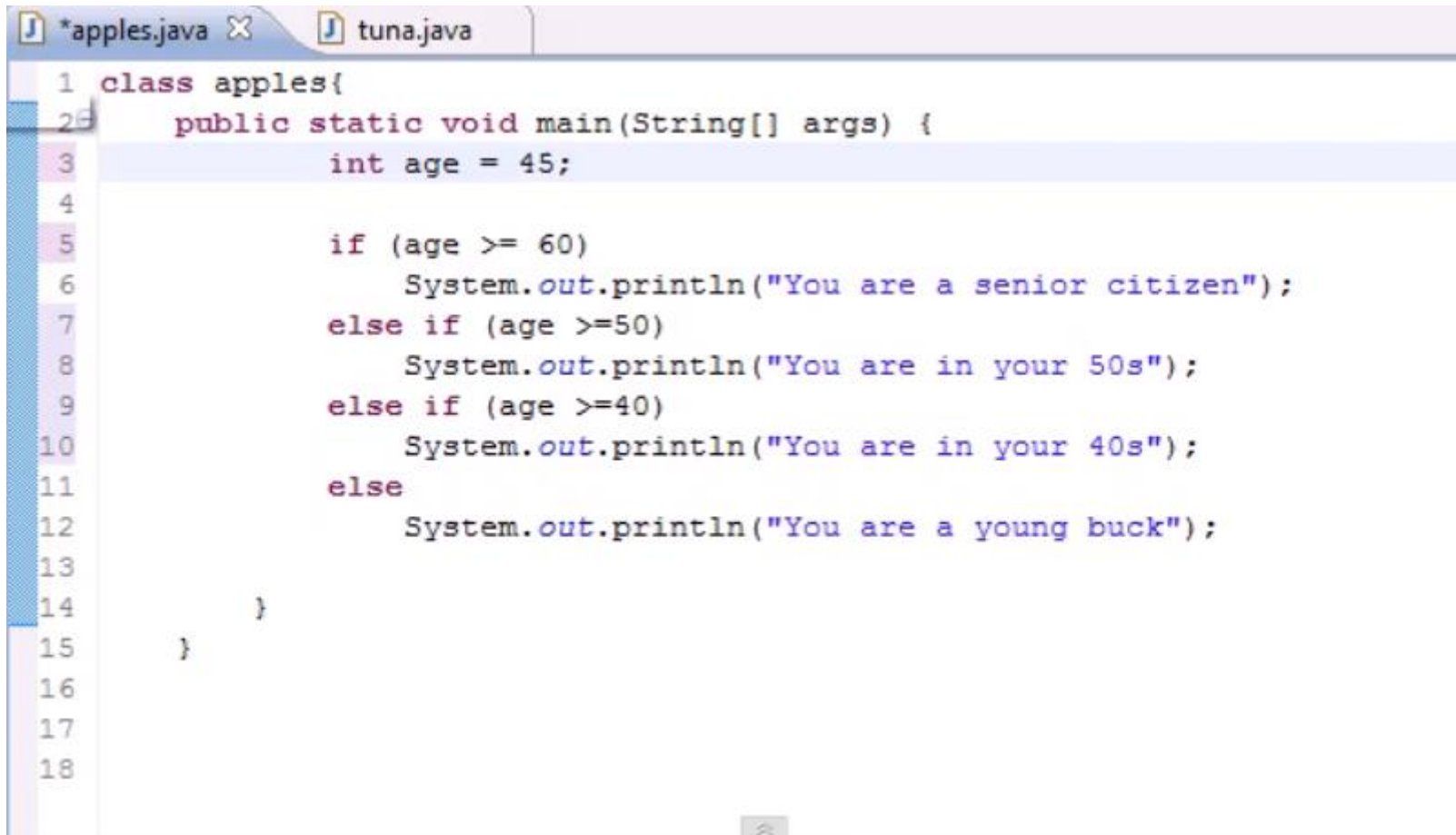


The screenshot shows an IDE's console window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application. The text in the console is as follows:

```
erminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 10, 2009 10:55:16 AM)  
ou are old  
ont worry you arent really thta old
```

The second line of output, "ou are old", is highlighted in blue. The third line, "ont worry you arent really thta old", is also highlighted in blue.

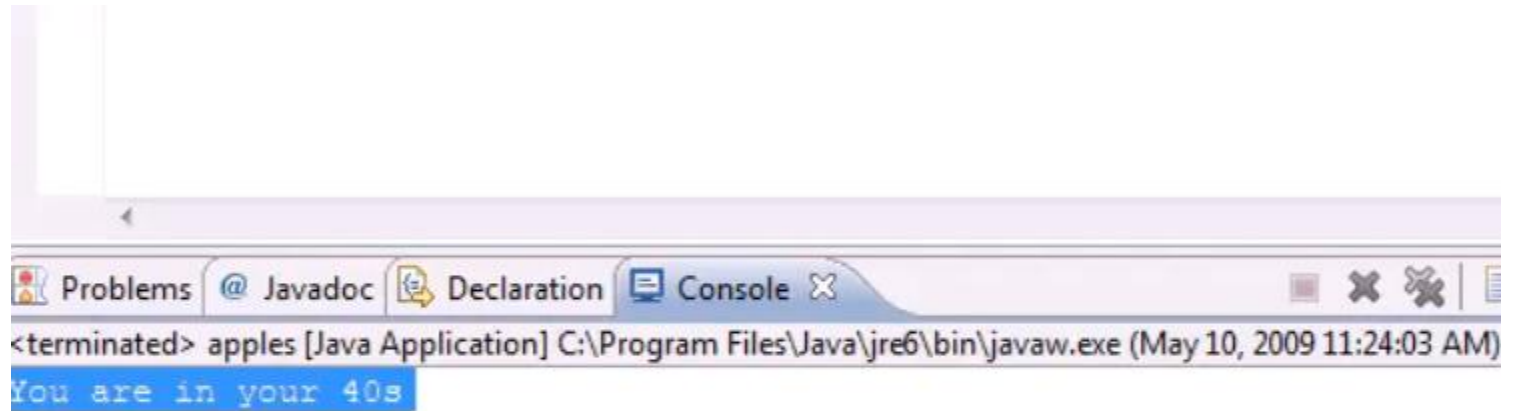
# Nested if in Java



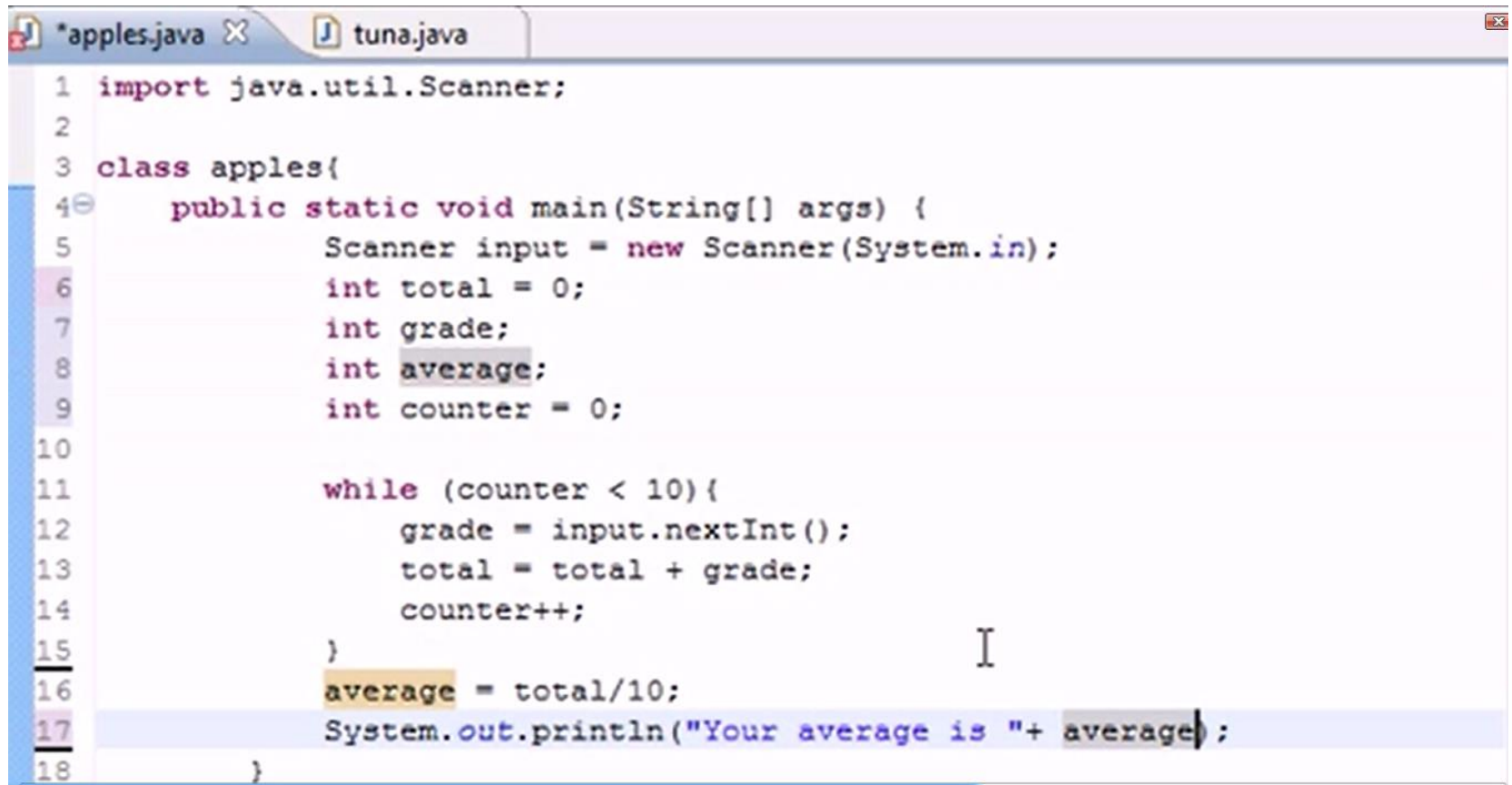
The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

```
1 class apples{
2     public static void main(String[] args) {
3         int age = 45;
4
5         if (age >= 60)
6             System.out.println("You are a senior citizen");
7         else if (age >=50)
8             System.out.println("You are in your 50s");
9         else if (age >=40)
10            System.out.println("You are in your 40s");
11        else
12            System.out.println("You are a young buck");
13
14    }
15 }
16
17
18
```

# View output



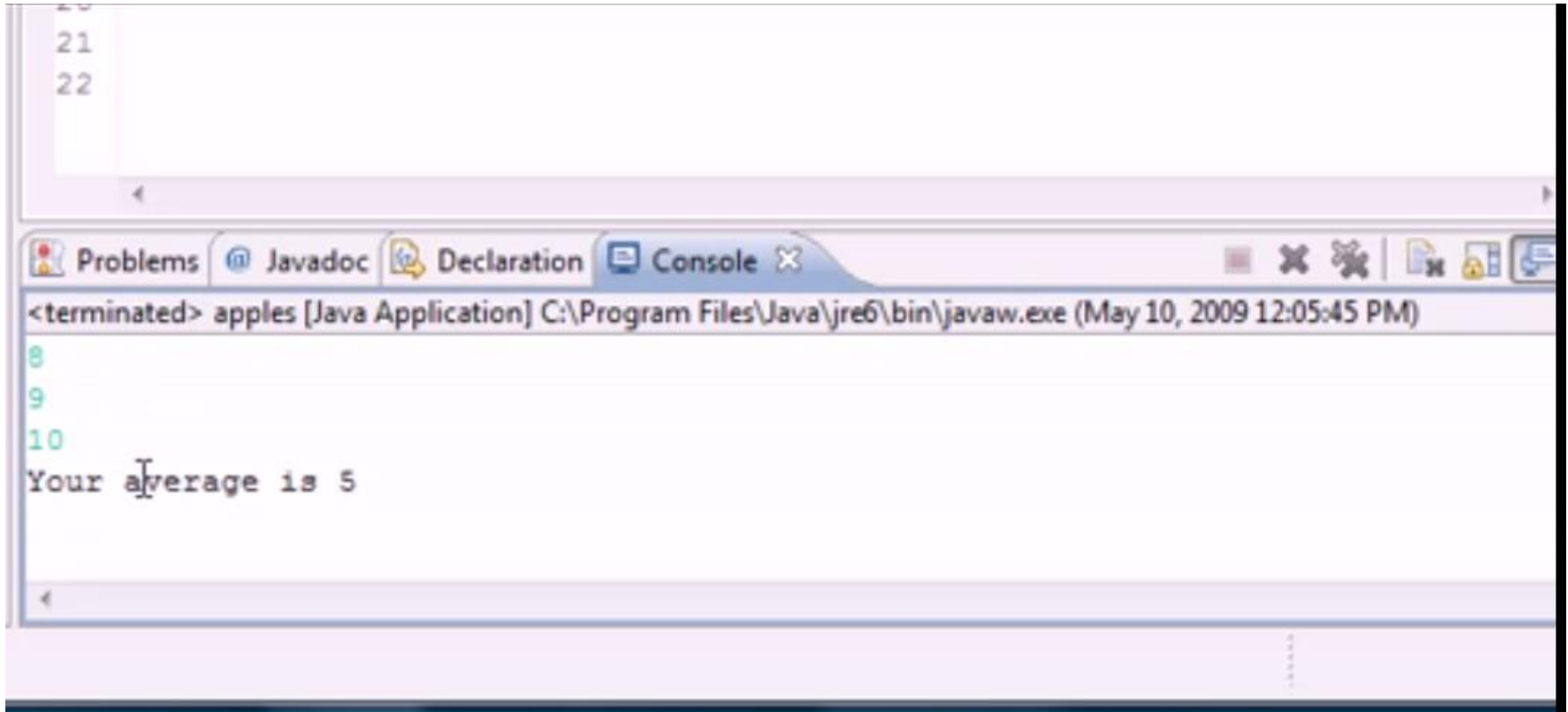
# Simple Average Program in Java

A screenshot of a Java IDE window with two tabs: 'apples.java' and 'tuna.java'. The 'apples.java' tab is active, showing a Java program to calculate the average of 10 numbers. The code is as follows:

```
1 import java.util.Scanner;
2
3 class apples{
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int total = 0;
7         int grade;
8         int average;
9         int counter = 0;
10
11         while (counter < 10){
12             grade = input.nextInt();
13             total = total + grade;
14             counter++;
15         }
16         average = total/10;
17         System.out.println("Your average is " + average);
18     }
```

The code is color-coded: keywords are in blue, identifiers and literals are in black, and string literals are in red. The variable 'average' is highlighted in yellow on lines 8, 16, and 17. A cursor is visible on line 15, and another cursor is on line 17 at the end of the string literal.

# View output



The screenshot shows an IDE window with a code editor at the top and a console window at the bottom. The code editor shows lines 21 and 22 of a file. The console window has tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, showing the output of a Java application named 'apples'. The output consists of three green numbers (8, 9, 10) followed by the text 'Your average is 5'.

```
21  
22
```

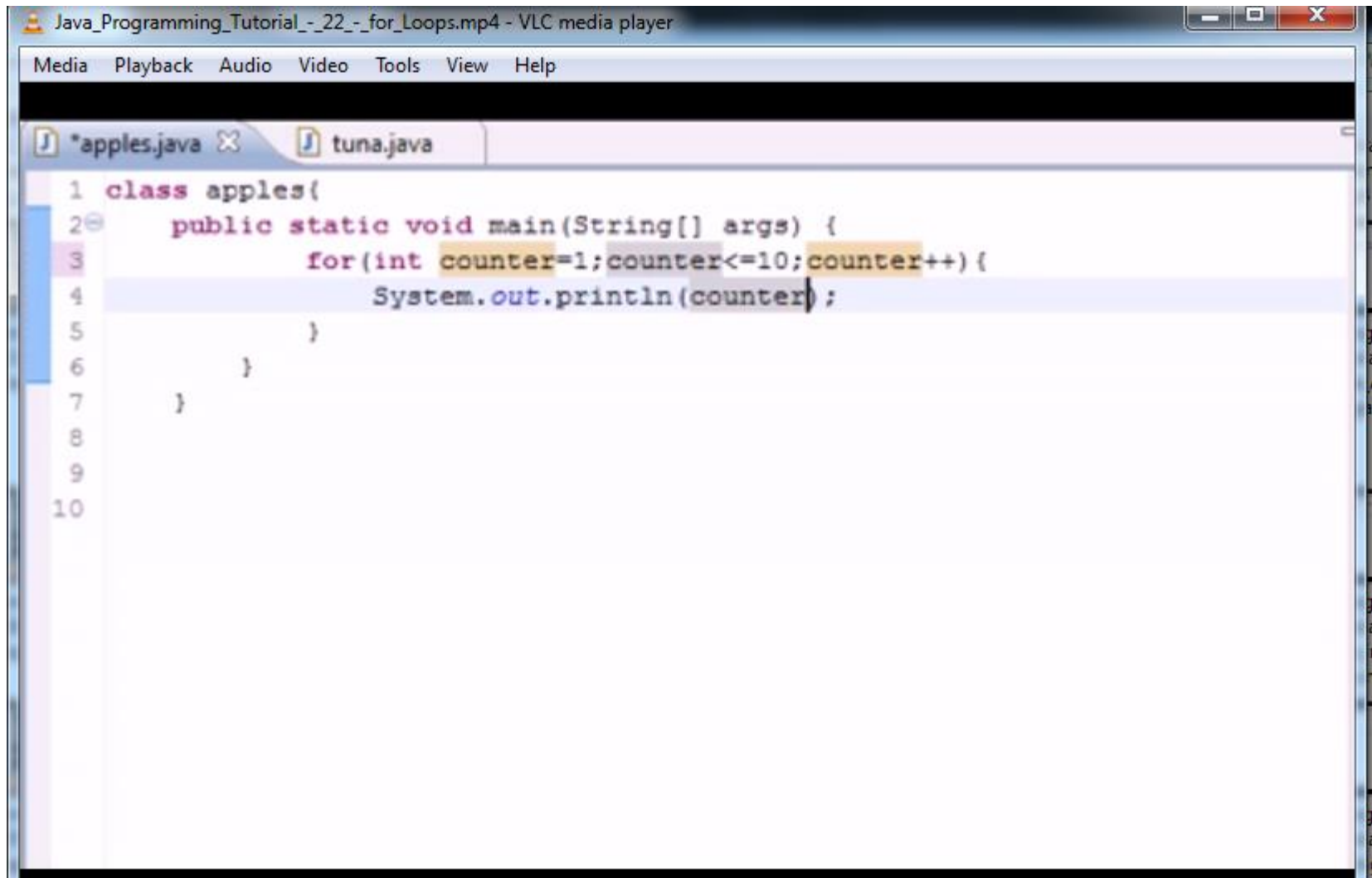
Problems Javadoc Declaration Console

<terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 10, 2009 12:05:45 PM)

8  
9  
10  
Your average is 5



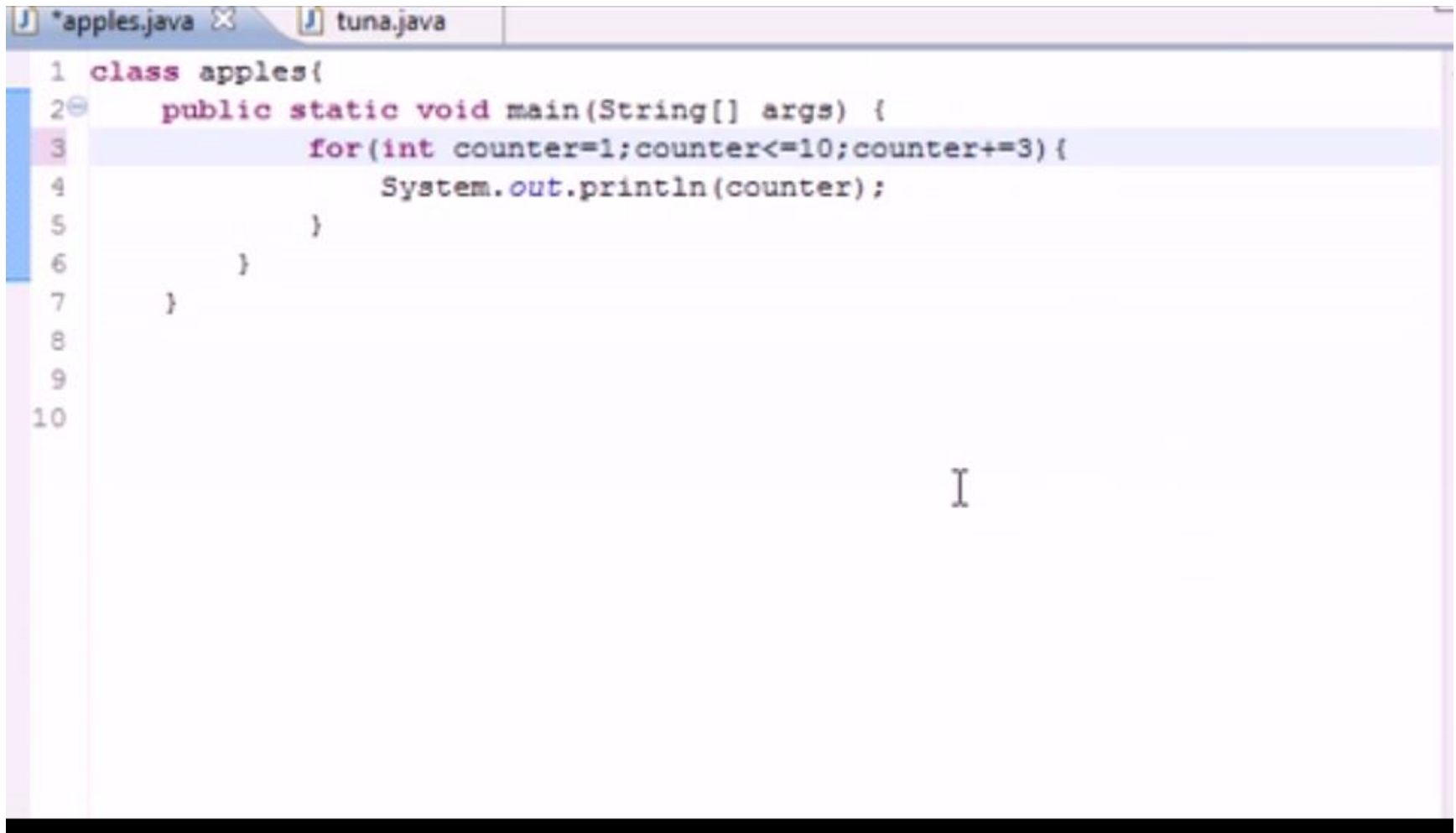
# For loop in Java



The screenshot shows a Java IDE window titled "Java\_Programming\_Tutorial\_-\_22\_-\_for\_Loops.mp4 - VLC media player". The menu bar includes "Media", "Playback", "Audio", "Video", "Tools", "View", and "Help". The IDE has two tabs: "\*apples.java" and "tuna.java". The code in the editor is as follows:

```
1 class apples{
2     public static void main(String[] args) {
3         for(int counter=1;counter<=10;counter++){
4             System.out.println(counter);
5         }
6     }
7 }
8
9
10
```

# For loop in Java

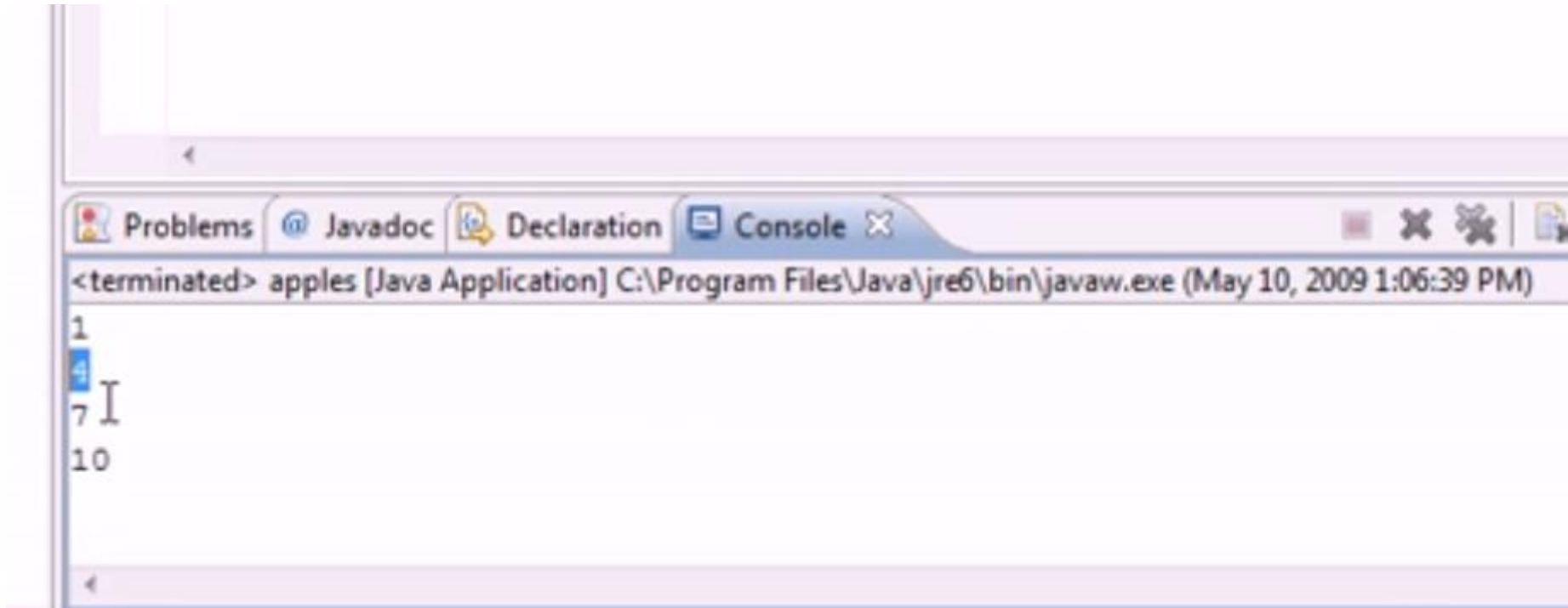


The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

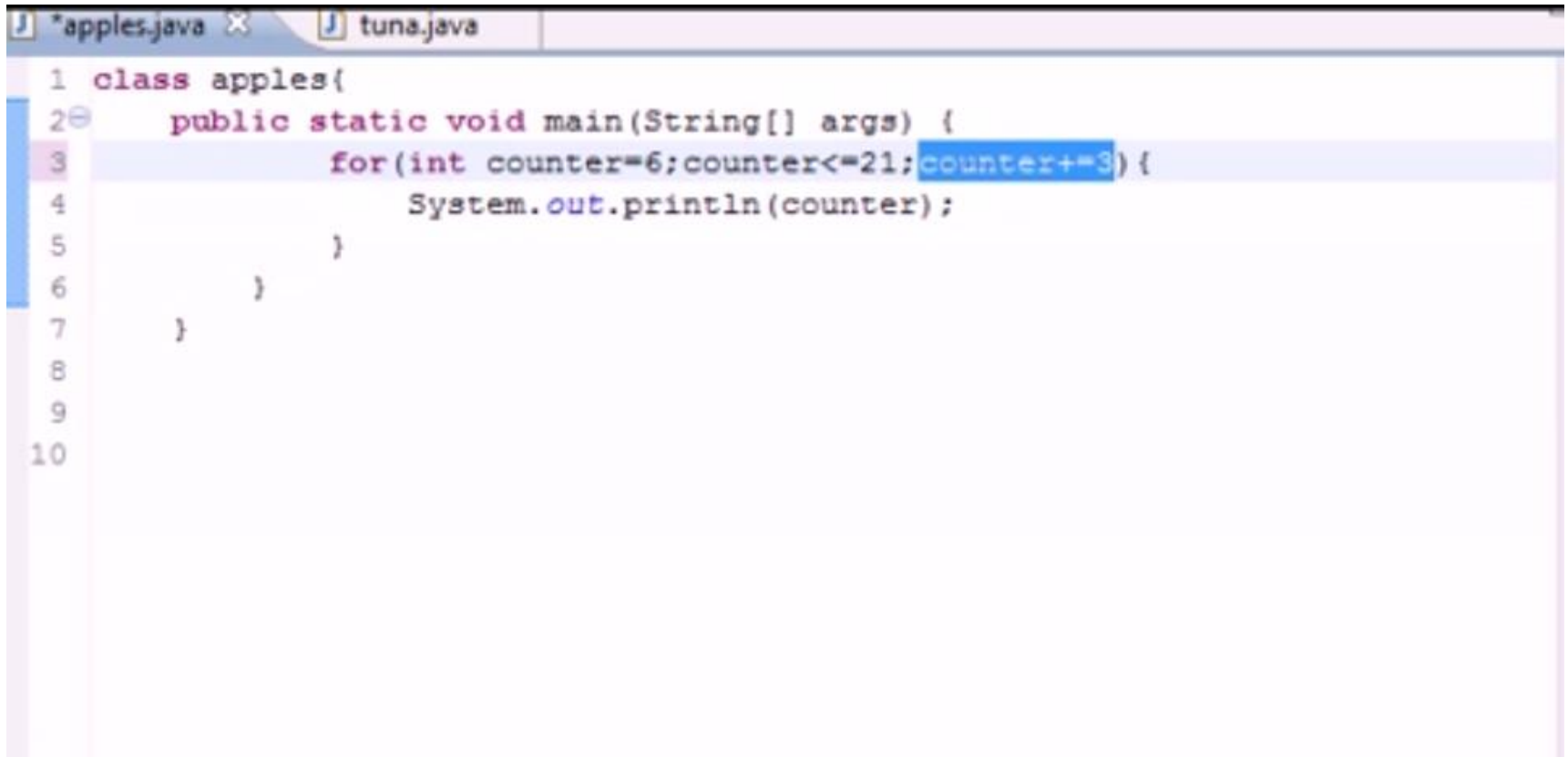
```
1 class apples{  
2     public static void main(String[] args) {  
3         for(int counter=1;counter<=10;counter+=3){  
4             System.out.println(counter);  
5         }  
6     }  
7 }  
8  
9  
10
```

The code defines a class `apples` with a `main` method. Inside the `main` method, a `for` loop is used to iterate over a range of values. The loop variable `counter` is initialized to 1, and the loop continues as long as `counter` is less than or equal to 10. The loop increments `counter` by 3 in each iteration. The `System.out.println(counter);` statement prints the value of `counter` to the console. The code is syntactically correct and will compile and run successfully.

# View output

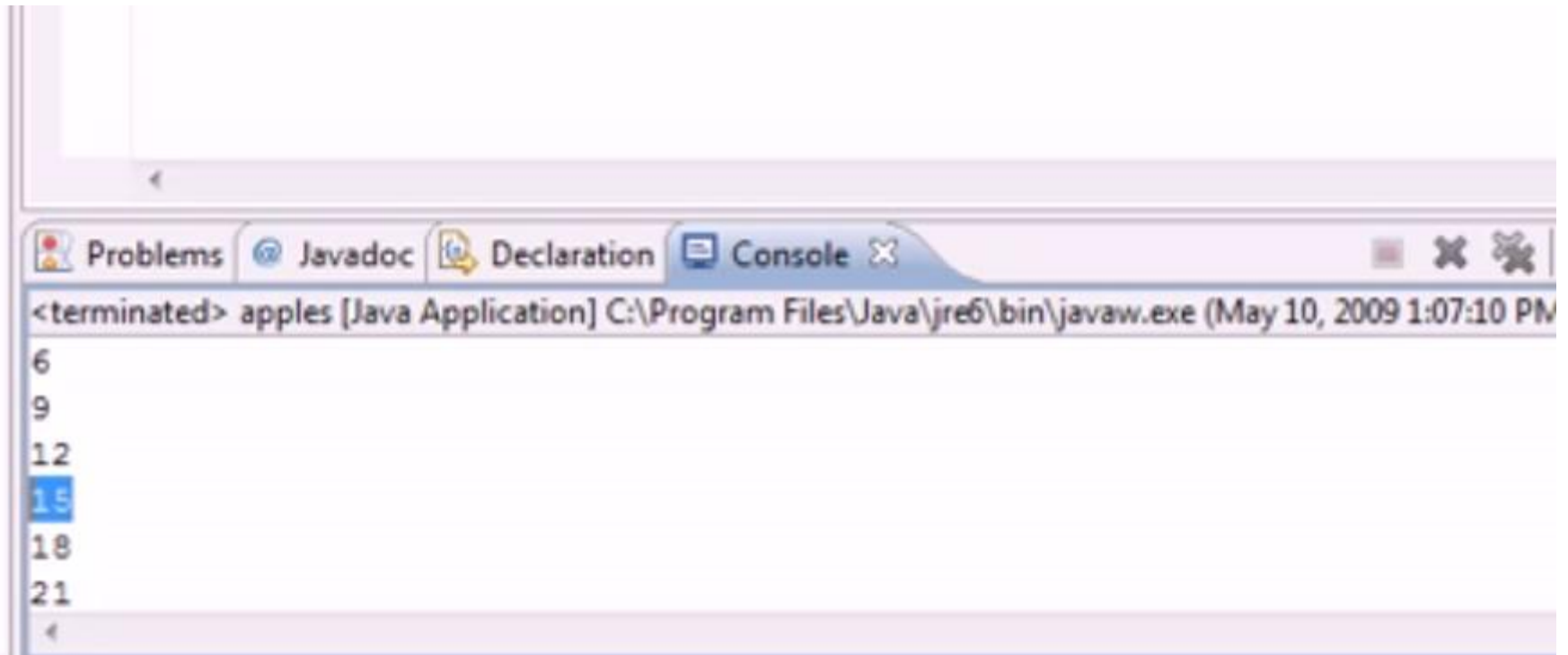


# Change increment section



```
1 class apples{
2     public static void main(String[] args) {
3         for(int counter=6;counter<=21;counter+=3){
4             System.out.println(counter);
5         }
6     }
7 }
8
9
10
```

# For loop in Java



The screenshot shows an IDE's console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console output is as follows:

```
<terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 10, 2009 1:07:10 PM)  
6  
9  
12  
15  
18  
21  
<
```

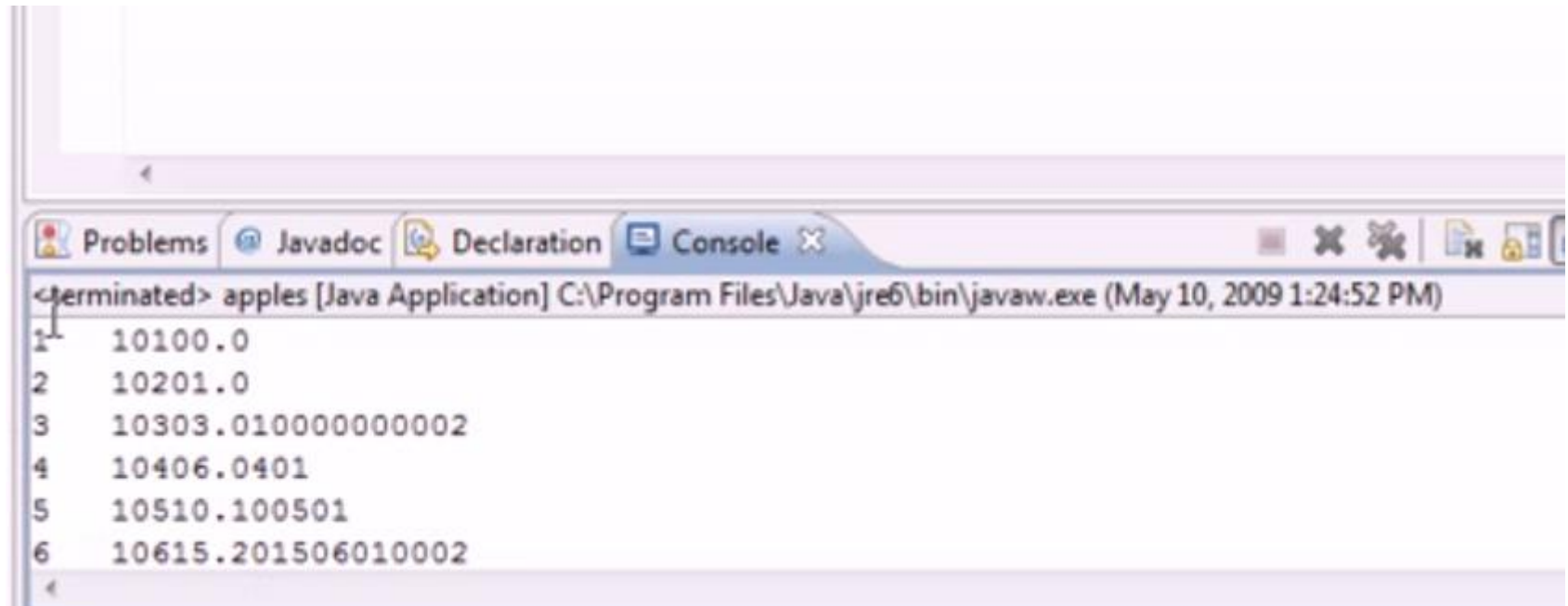
The number 15 is highlighted in blue in the original image.

# Compound Interest in Java

```
*apples.java X  tuna.java

1 class apples{
2     public static void main(String[] args) {
3         double amount;
4         double principal = 10000;
5         double rate = .01;
6
7         for(int day=1;day<=20;day++){
8             amount=principal*Math.pow(1 + rate, day);
9             System.out.println(day + "    "+ amount);
10        }
11    }
12 }
13
14
15
```

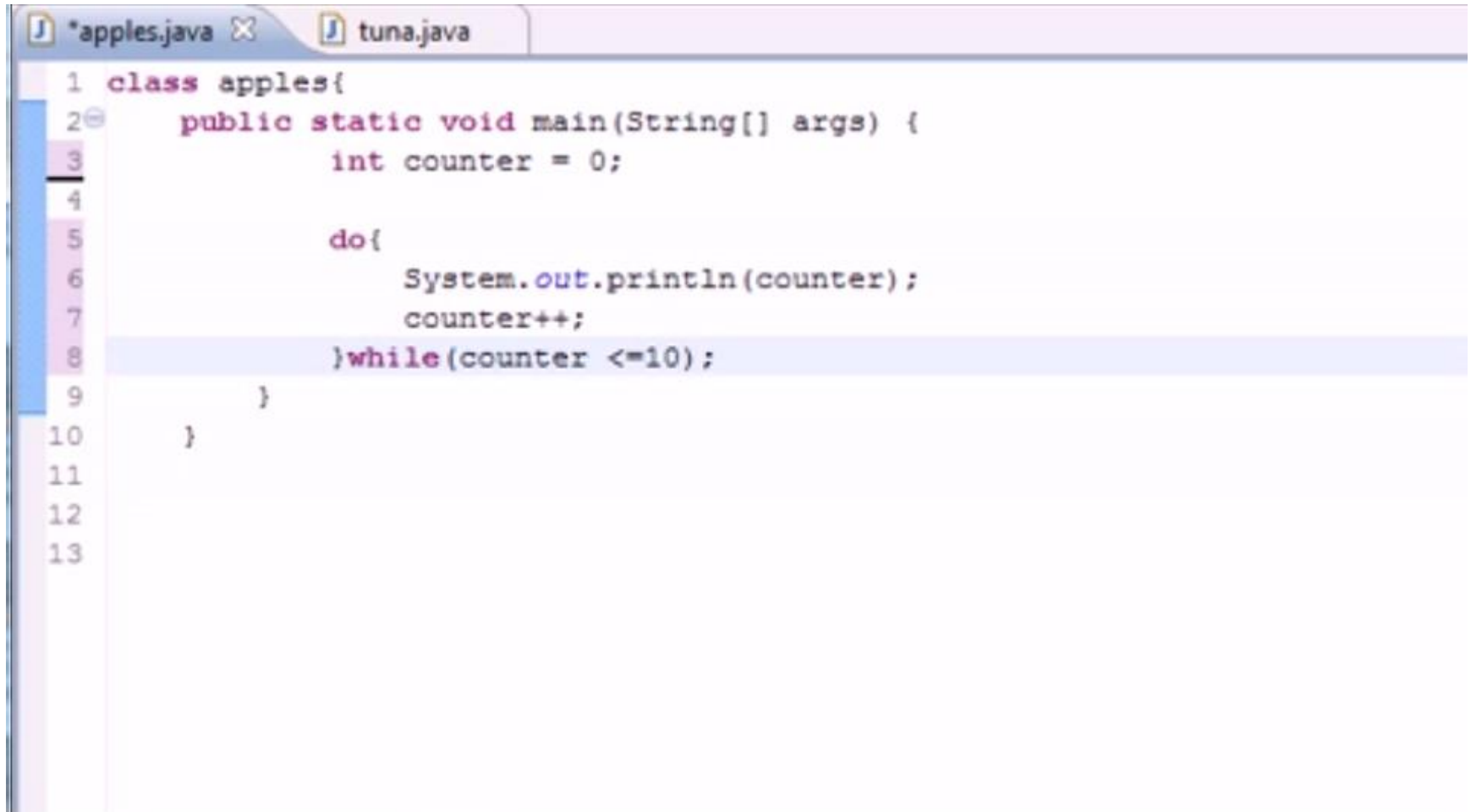
# Compound Interest in Java



The screenshot shows a Java IDE window with the 'Console' tab selected. The console output displays the results of a compound interest calculation for an application named 'apples'. The output consists of six lines, each representing a time step from 1 to 6. Each line shows a numerical value that increases over time, representing the accumulated amount with compound interest. The values are: 10100.0, 10201.0, 10303.010000000002, 10406.0401, 10510.100501, and 10615.201506010002.

```
<terminated> apples [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 10, 2009 1:24:52 PM)
1 10100.0
2 10201.0
3 10303.010000000002
4 10406.0401
5 10510.100501
6 10615.201506010002
```

# Do-while in Java

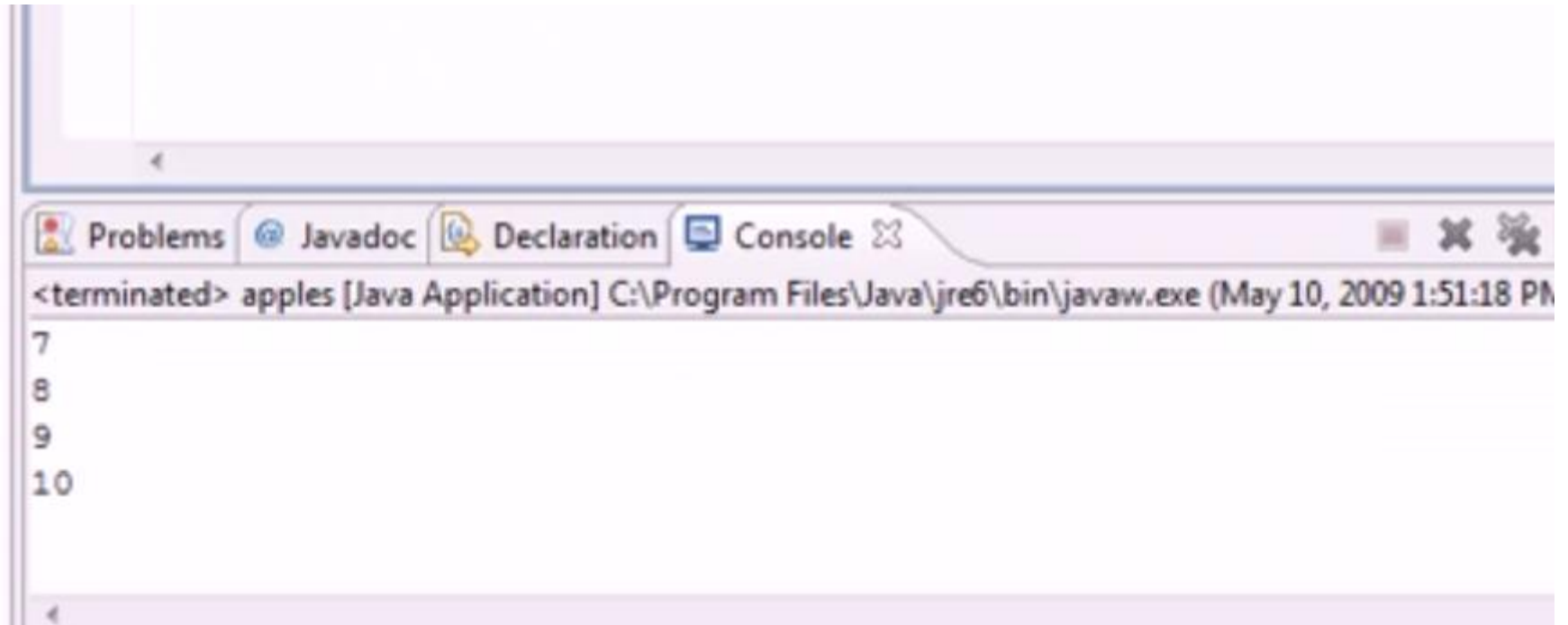


The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

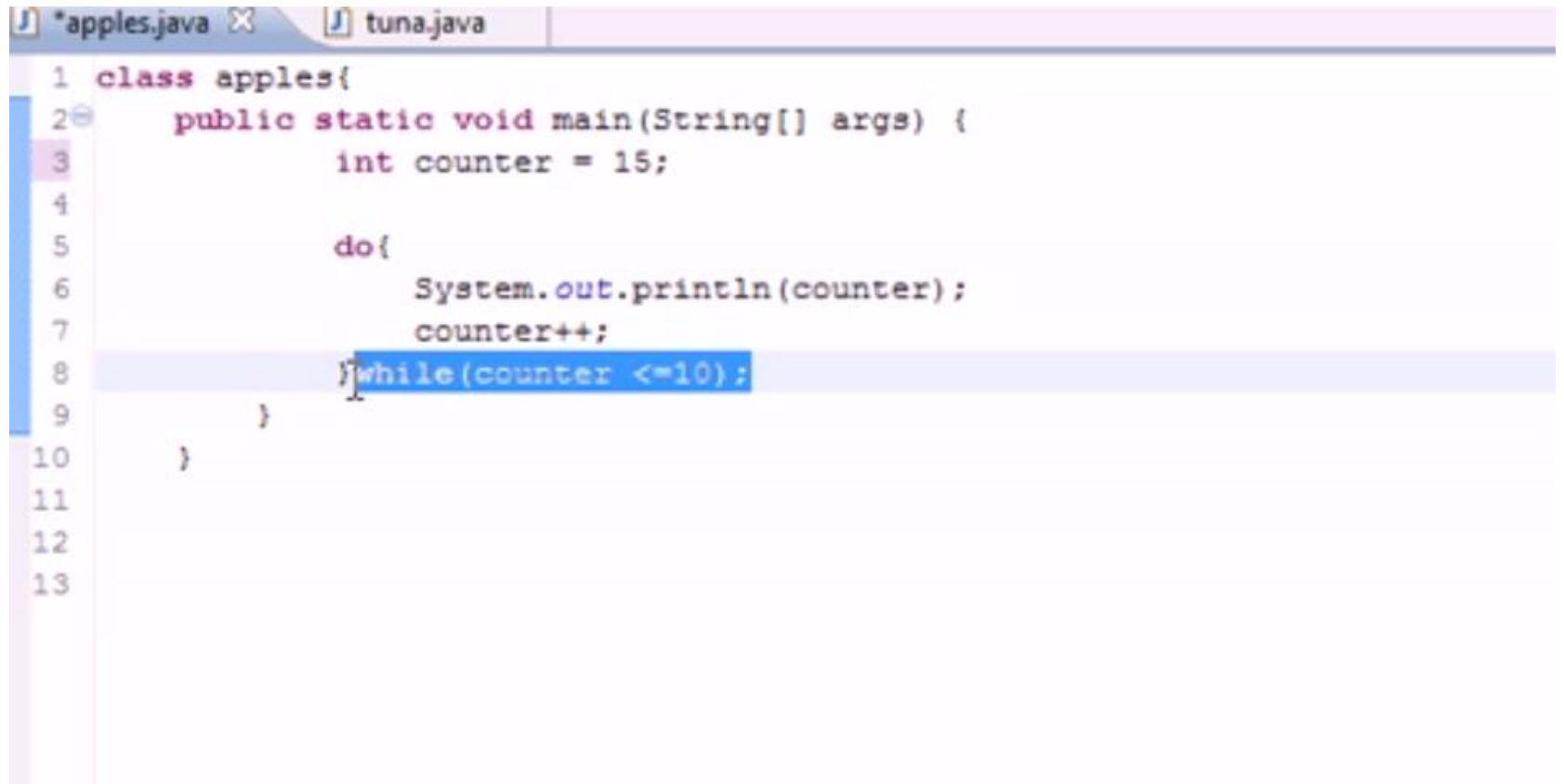
```
1 class apples{  
2     public static void main(String[] args) {  
3         int counter = 0;  
4  
5         do{  
6             System.out.println(counter);  
7             counter++;  
8         }while(counter <=10);  
9     }  
10 }  
11  
12  
13
```



# View output



# Change counter value

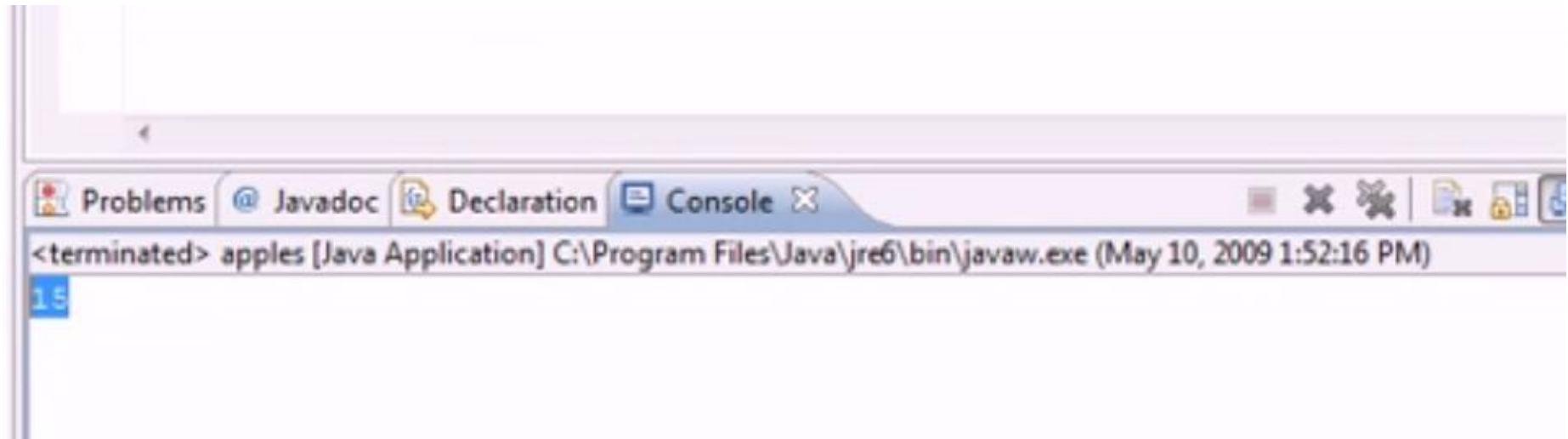


The screenshot shows a Java code editor with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

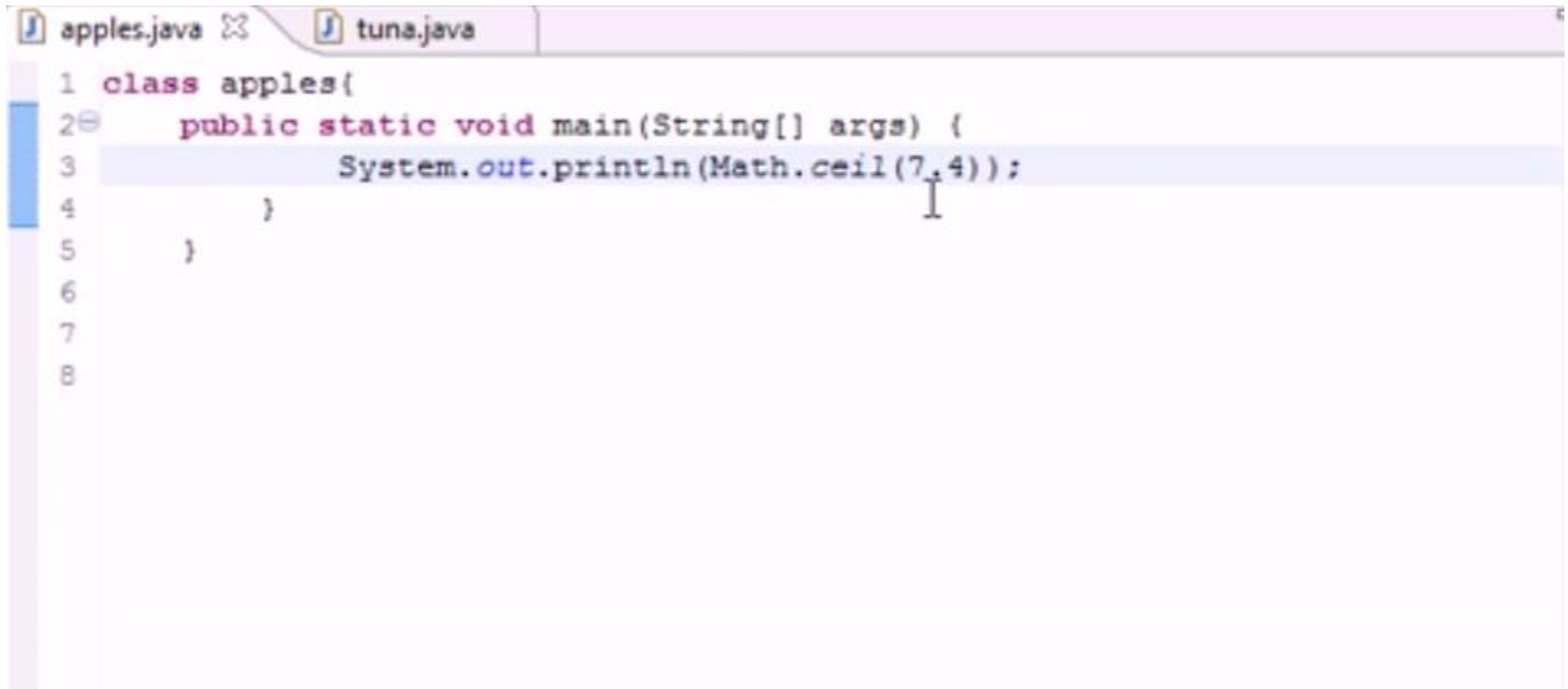
```
1 class apples{
2     public static void main(String[] args) {
3         int counter = 15;
4
5         do{
6             System.out.println(counter);
7             counter++;
8         }while(counter <=10);
9     }
10 }
11
12
13
```

The code defines a class `apples` with a `main` method. Inside the `main` method, a variable `counter` is initialized to 15. A `do-while` loop is used to print the value of `counter` and increment it by 1, as long as `counter` is less than or equal to 10. The loop body is highlighted in blue, and the `while` condition is highlighted in light blue. The line numbers 1 through 13 are visible on the left side of the editor.

# View output



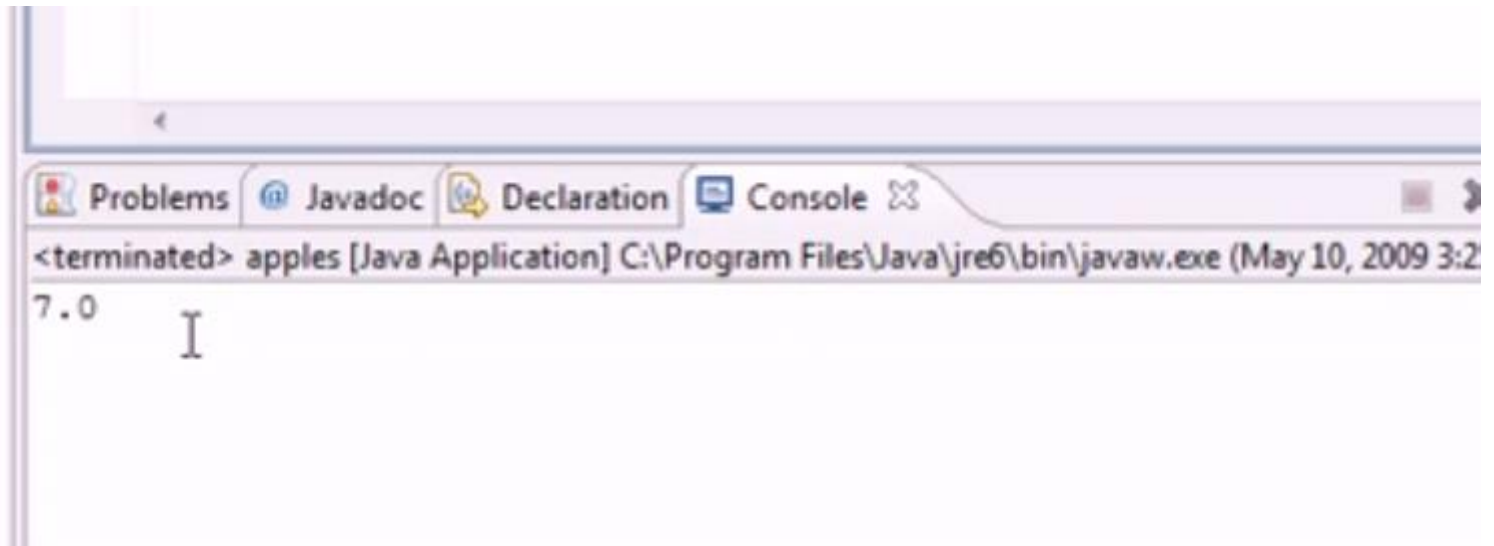
# Math Class Methods in java: ceil



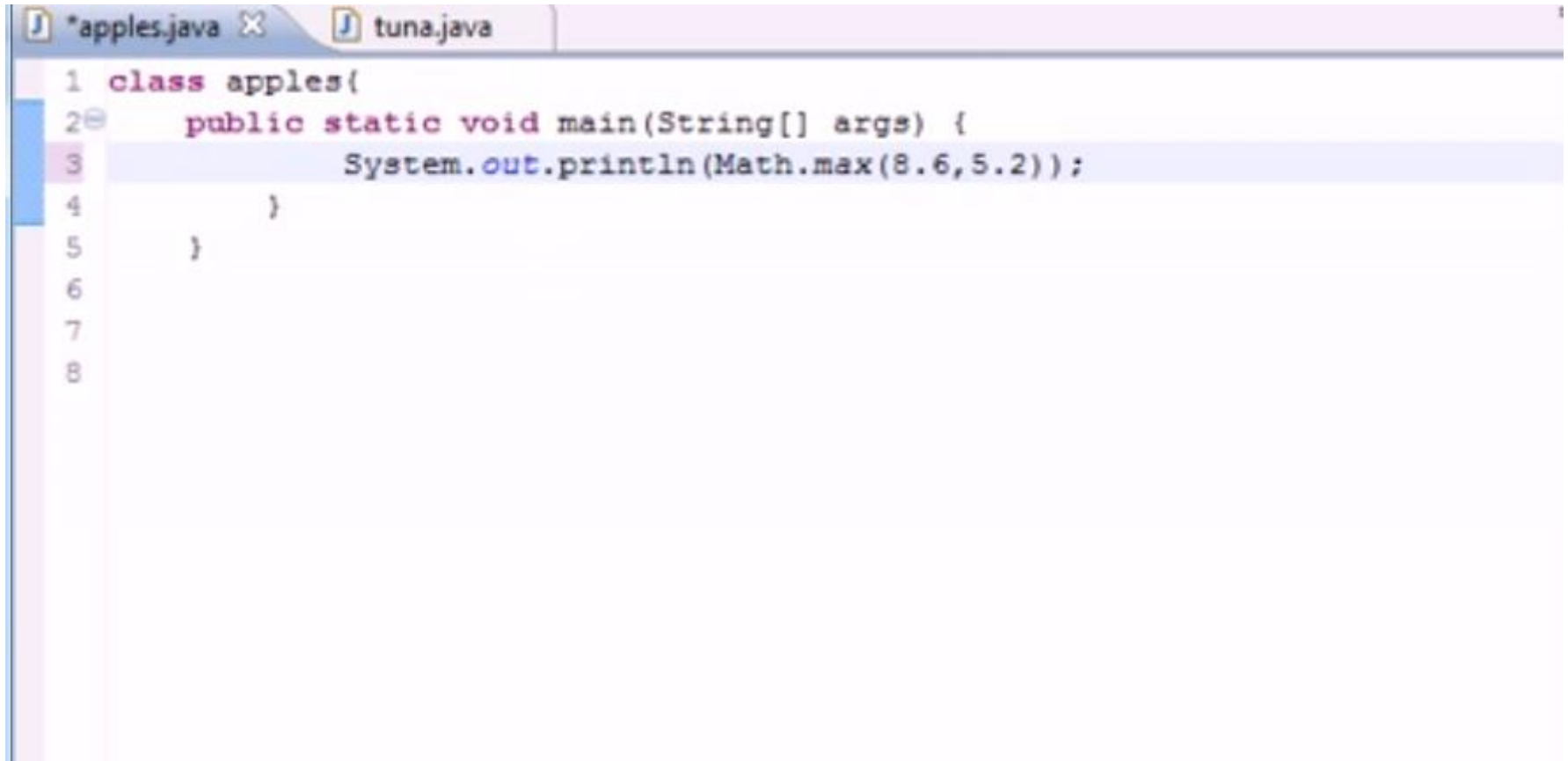
The screenshot shows a Java IDE with two tabs: 'apples.java' and 'tuna.java'. The 'apples.java' tab is active, displaying a Java class named 'apples'. The class contains a 'main' method that prints the result of 'Math.ceil(7.4)'. The code is as follows:

```
1 class apples{  
2     public static void main(String[] args) {  
3         System.out.println(Math.ceil(7.4));  
4     }  
5 }  
6  
7  
8
```

# View output



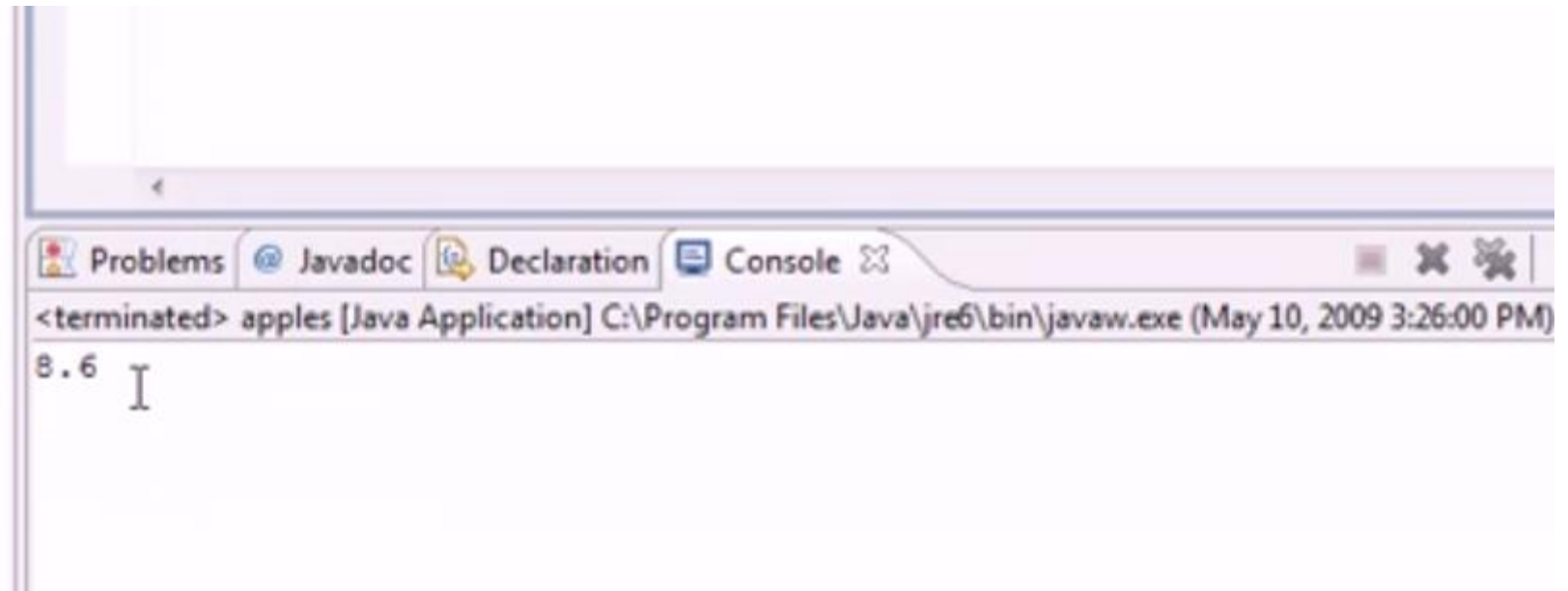
# Math Class Methods in Java: max



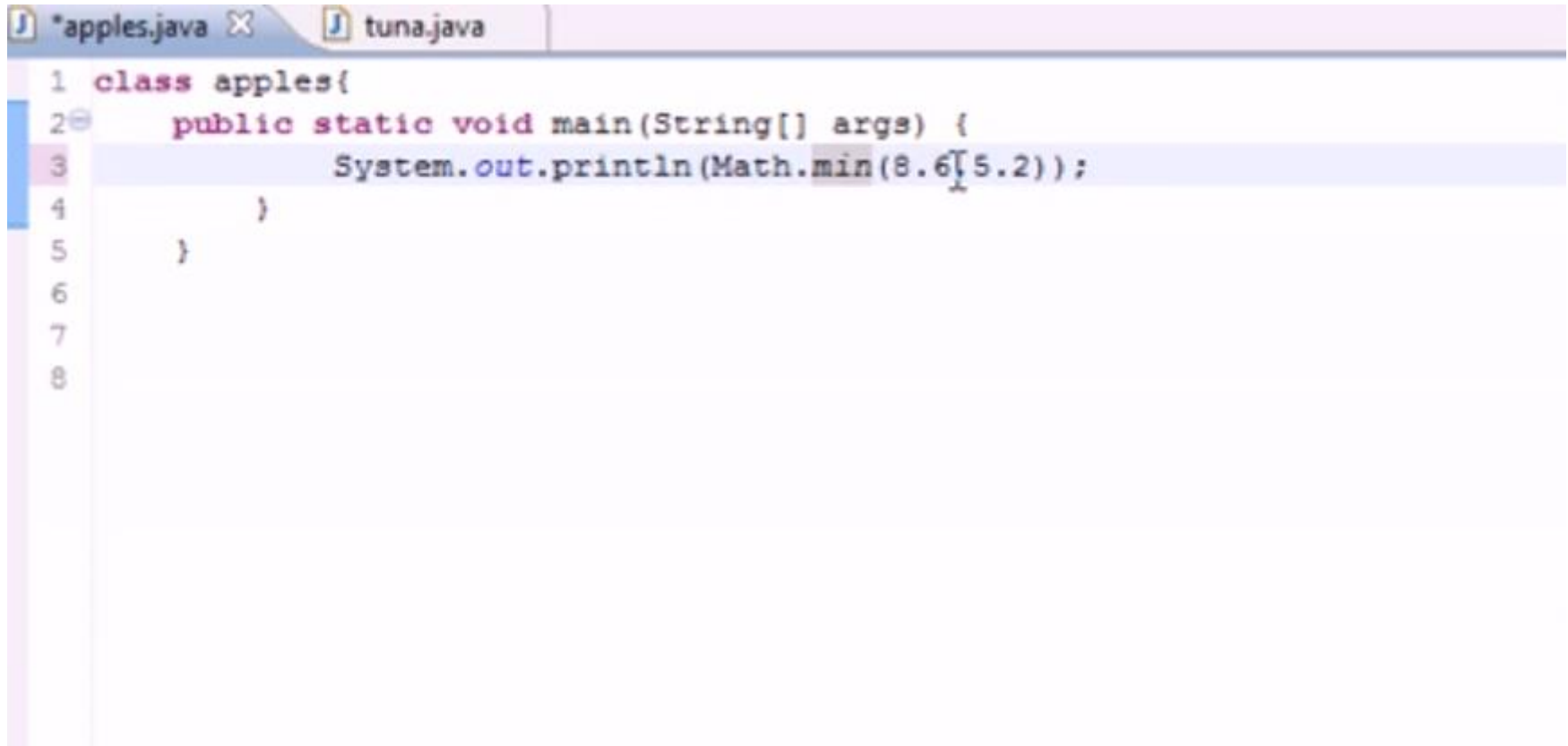
The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `apples.java` tab is active, displaying the following code:

```
1 class apples{  
2     public static void main(String[] args) {  
3         System.out.println(Math.max(8.6, 5.2));  
4     }  
5 }  
6  
7  
8
```

# View output



# Math Class Methods in Java: min



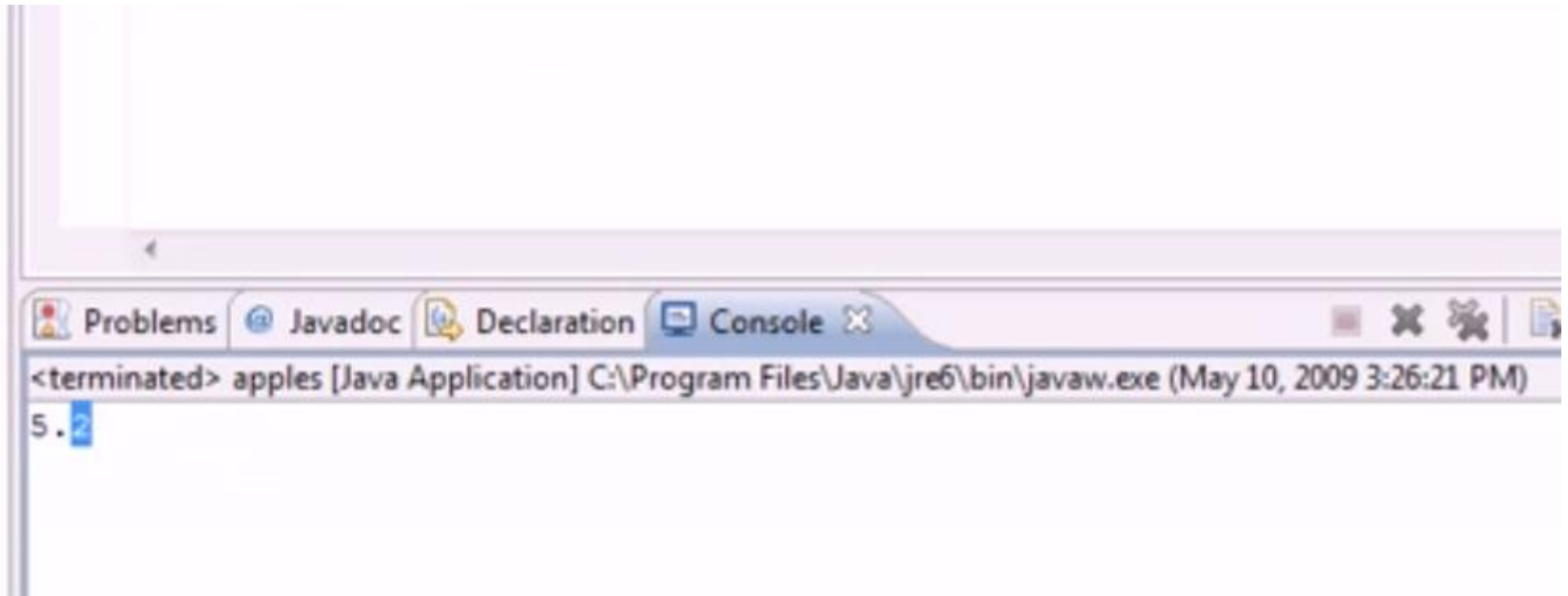
The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

```
1 class apples{  
2     public static void main(String[] args) {  
3         System.out.println(Math.min(8.6, 5.2));  
4     }  
5 }  
6  
7  
8
```

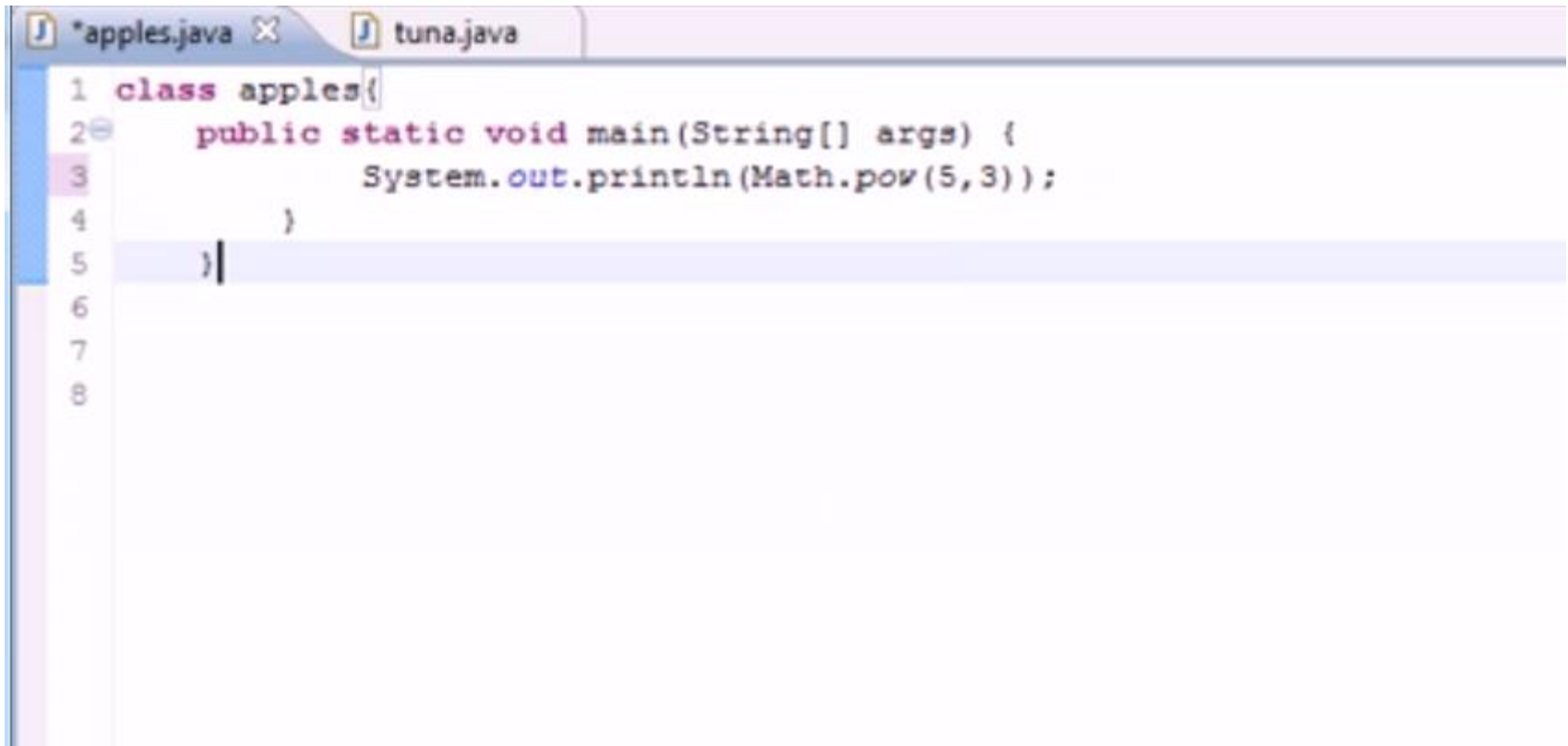
The code defines a class named `apples` with a `main` method. Inside the `main` method, it prints the result of `Math.min(8.6, 5.2)` to the console. The `min` method is highlighted in the original image.



# View output



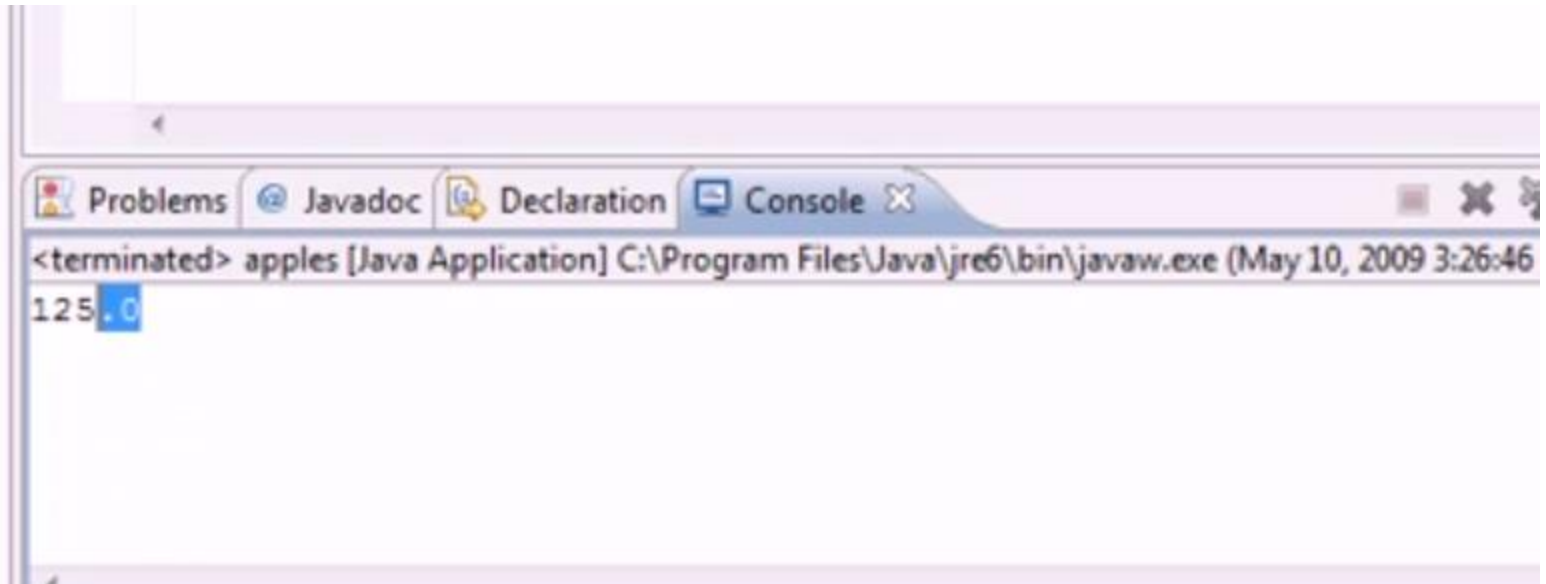
# Math Class Methods in Java: pow



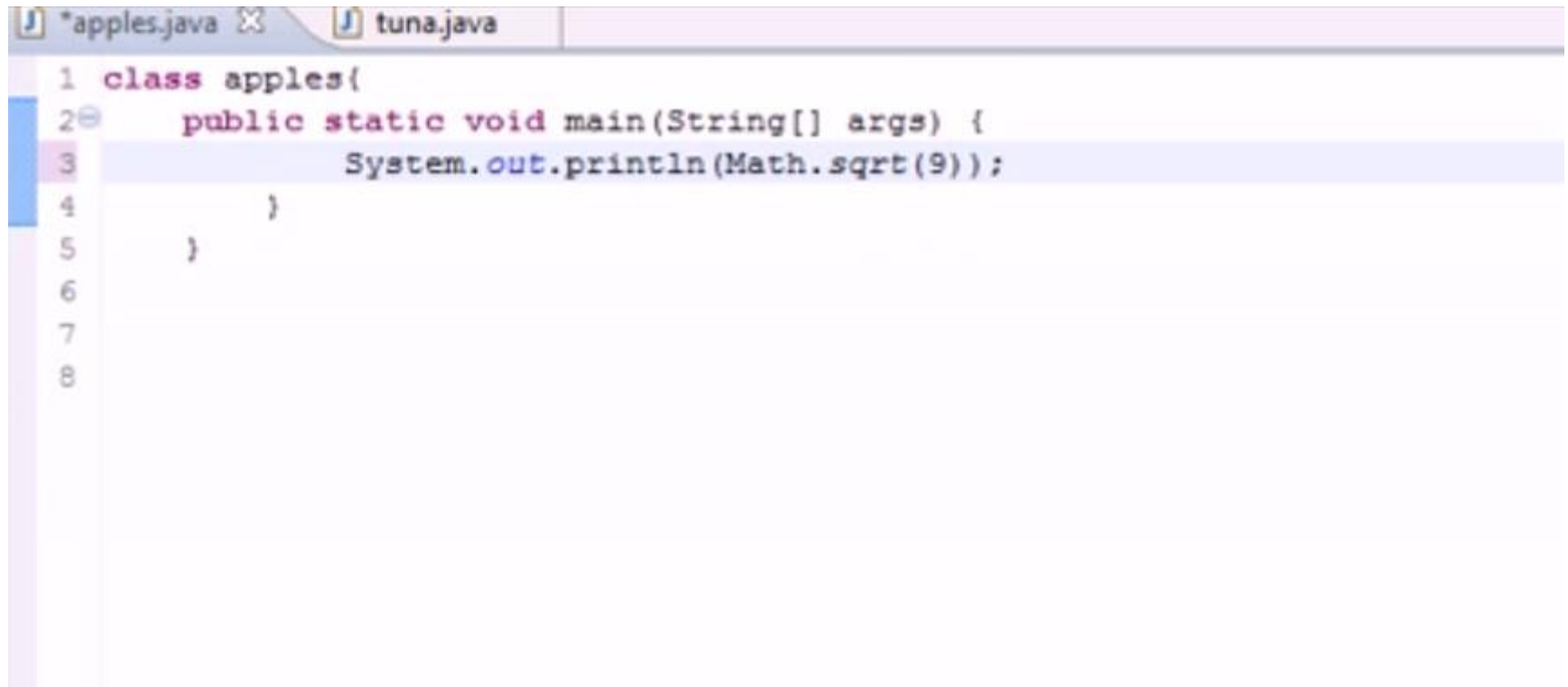
The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

```
1 class apples{  
2     public static void main(String[] args) {  
3         System.out.println(Math.pow(5,3));  
4     }  
5 }  
6  
7  
8
```

# View output



# Math Class Methods in Java: sqrt



The screenshot shows a Java IDE with two tabs: `*apples.java` and `tuna.java`. The `*apples.java` tab is active, displaying the following code:

```
1 class apples{  
2     public static void main(String[] args) {  
3         System.out.println(Math.sqrt(9));  
4     }  
5 }  
6  
7  
8
```

# View output

