CS 134:
Introduction to Java

Slide content based on http://www.cs.cmu.edu/~mjs/courses/121-F14-W/Java4Python.pdf

Announcements & Logistics

- **Lab 9 Boggle :** Due tonight/tomorrow @ 11pm
  - Lots of office hours
  - Come talk to us if you have questions!
- **HW 9** available today, due Mon 5/9 @ 11pm
  - Covers "advanced" topics from recent lectures
- **Lab 10 Selection Sort in Java** (next Mon/Tue)
  - No pre-lab work; hope most of you will start and finish during your lab session
- **Final exam reminder: Sunday, May 22 @ 9:30 AM**
- Course evals next Friday 5/13 (bring a laptop to class if possible)

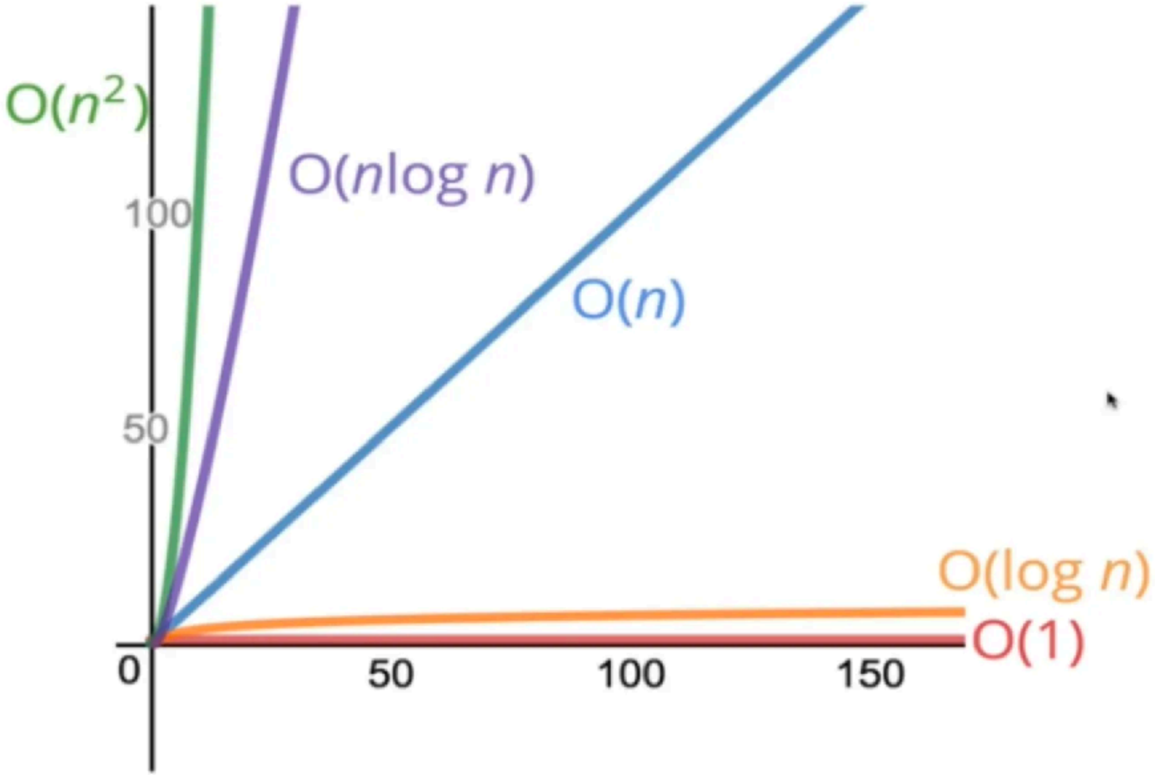
Do You Have Any Questions?

May the 4th Be With You

- Working on partner labs be like — hopefully none of you had to resort to `git push --force`!

Last Time

- Briefly reviewed searching algorithms:
    - $O(\log n)$: binary search runtime in a sorted array-based list
    - $O(n)$: linear searching runtime in an unsorted list
- Discussed two classic sorting algorithms:
    - $O(n \log n)$: merge sort runtime
    - $O(n^2)$: selection sort runtime
- What about (extra) space for sorting?
    - $O(n)$: naive merge sort
    - $O(1)$: selection sort
- Time-space tradeoff!



Today

- Begin discussing **Java**
  - Discuss how to **run programs in Java**
  - Learn about Java **syntax**
  - Take a closer look at **data types** in Java
- Goals of next 4-5 lectures:
  - Understand the key similarities and differences between Python and other programming languages (Java)
  - **Review basic features of Python** in preparation for final exam
  - Gain confidence in our programming abilities
  - Help ease the transition to CS 136 (and beyond!)

Python vs. Java

Python

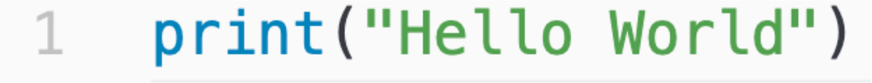
- Powerful language used by many programmers
- Features for making common programming tasks relatively simple
- Can run programs as scripts or interactively
- Dynamically typed: Run-time error when variables are used incorrectly
- Good fit for teaching programming to new computer scientists

Java

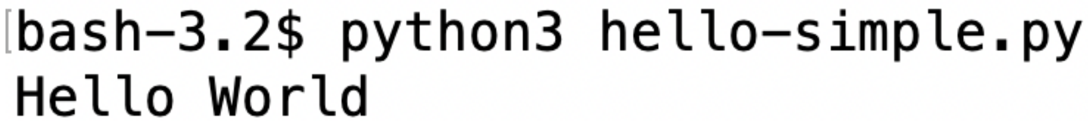
- Powerful language used by many programmers
- Features for building large-scale systems design
- Must be "compiled" and run from terminal
- Statically typed: compile-time error when variables are used incorrectly
- Good fit for large software projects, but steep learning curve

Hello, World!

Python in Week 1:

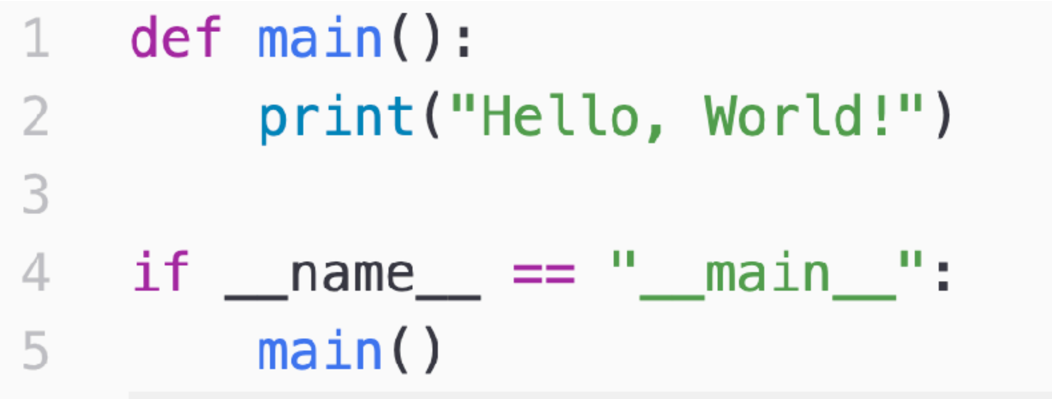


```python
print("Hello World")
```

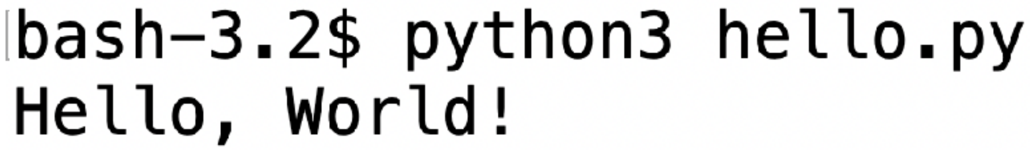


```
bash-3.2$ python3 hello-simple.py
Hello World
```

Python in Week 11:

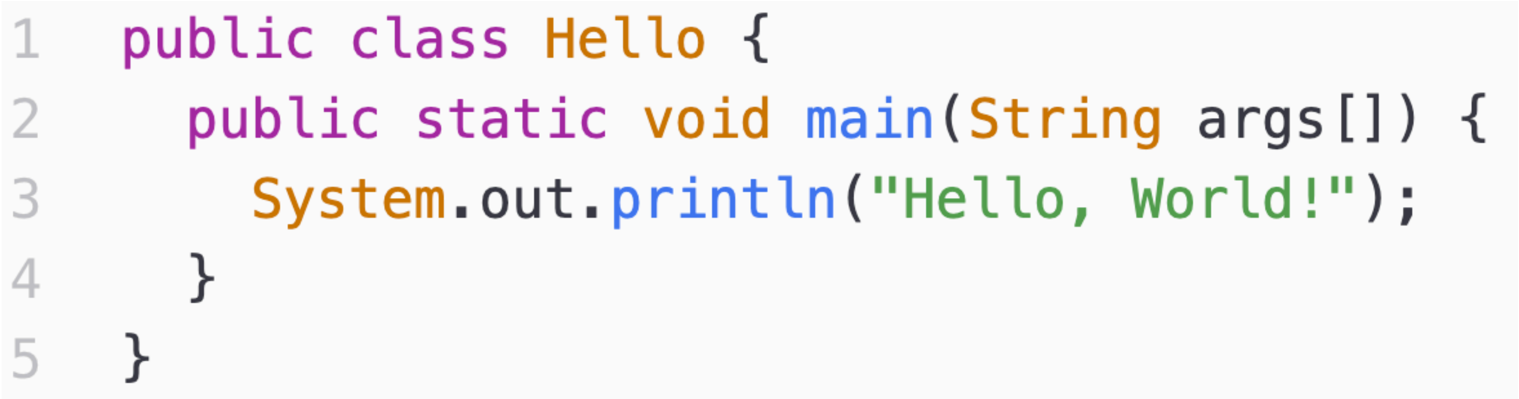


```python
def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

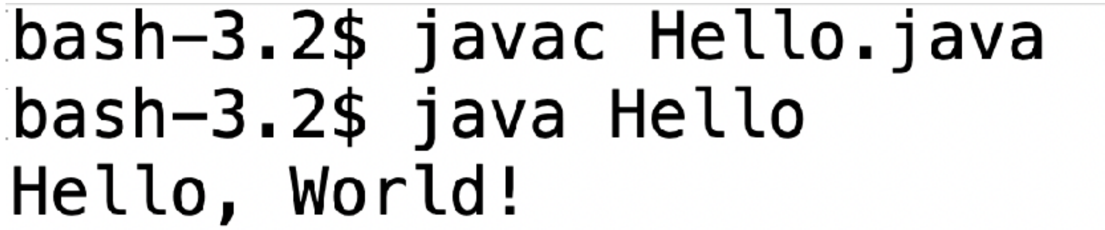


```
bash-3.2$ python3 hello.py
Hello, World!
```

Java:



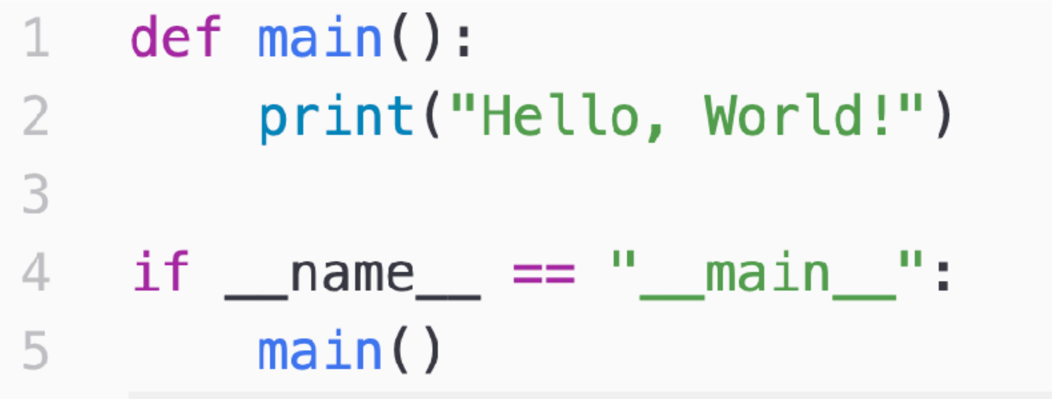
```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```



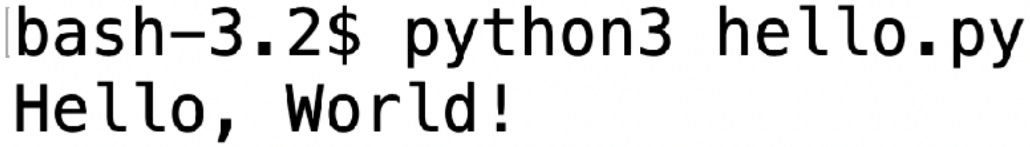
```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
```

Hello, World!

Python:

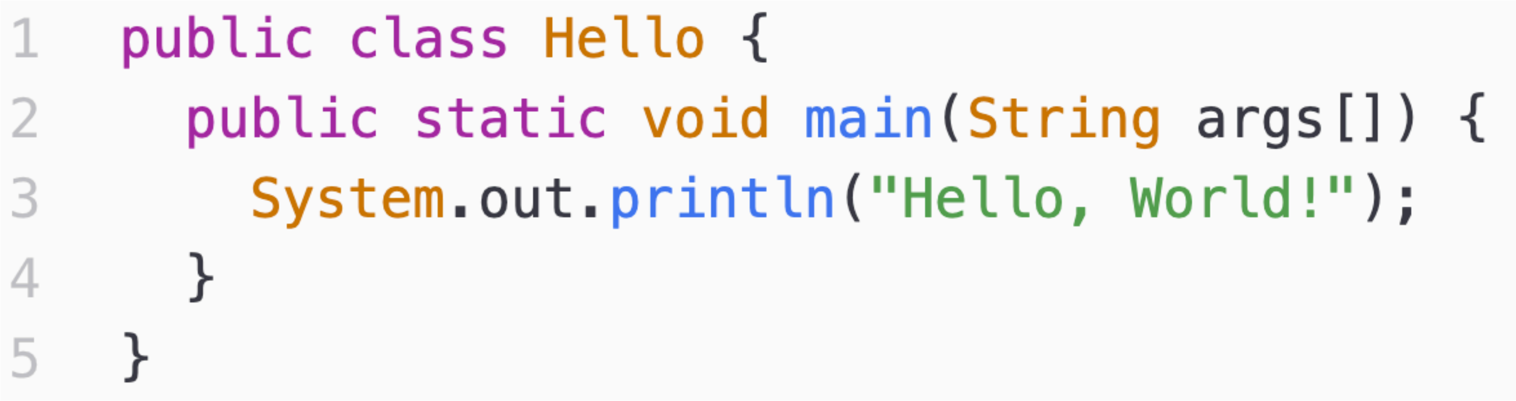


```python
def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

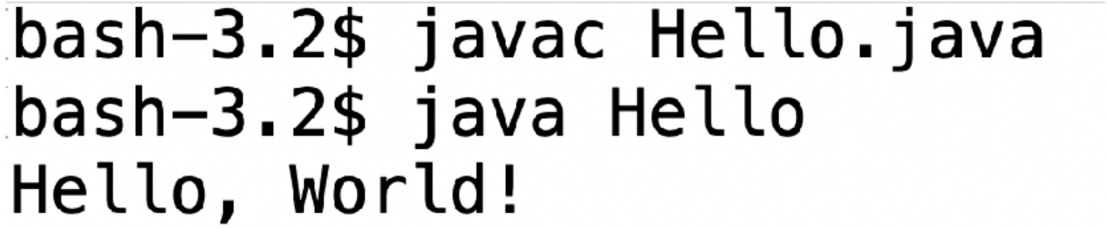


```
bash-3.2$ python3 hello.py
Hello, World!
```

Java:



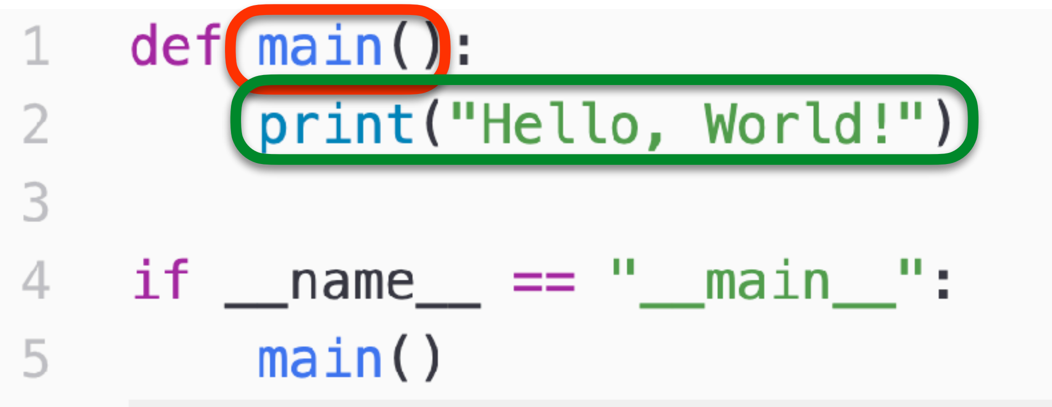
```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```



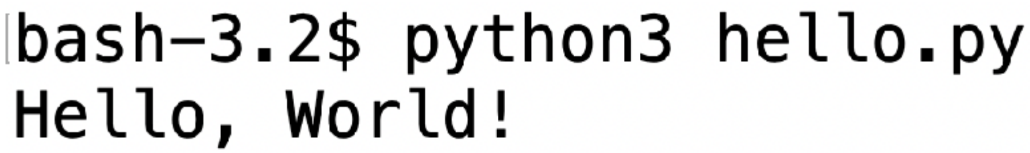
```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
```

Hello, World!

Python:

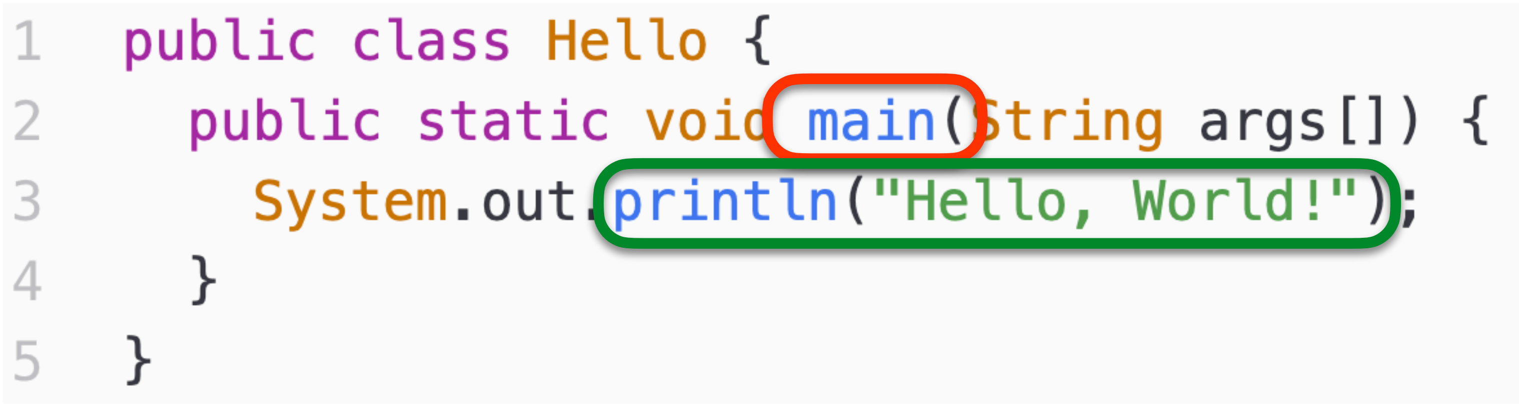


```python
def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

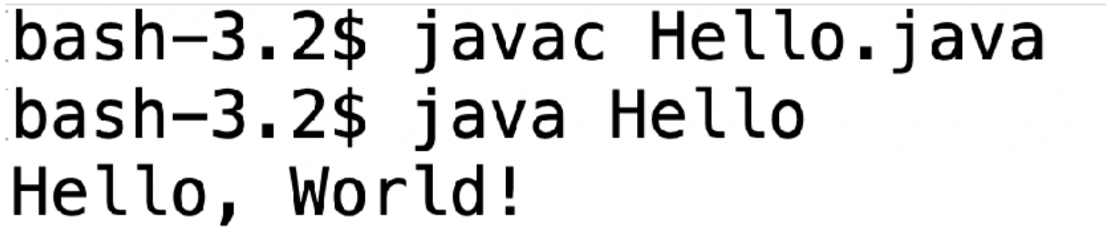


```
bash-3.2$ python3 hello.py
Hello, World!
```

Java:



```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```



```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
```

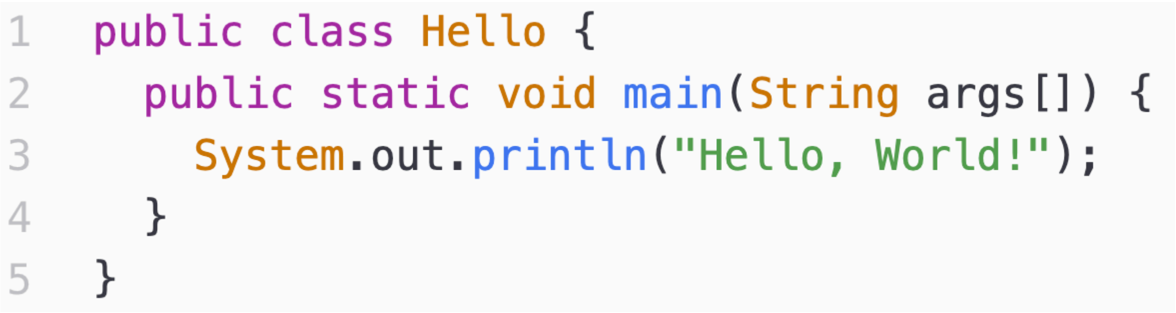
Running Our Code

- **Python** is an **interpreted** language
  - The Python ***interpreter*** runs through our code line by line and executes each command
  - Other interpreted languages: PHP, R, Ruby, and JavaScript
- **Java** is a **compiled** language*
  - The Java ***compiler*** converts our code into machine code that the processor can execute
  - Compiled languages require code to be ***manually compiled*** before execution
  - Other compiled languages: C, C++, Haskell, Rust, and Go
- Interpreted languages were once significantly slower than compiled languages. But that gap is shrinking.

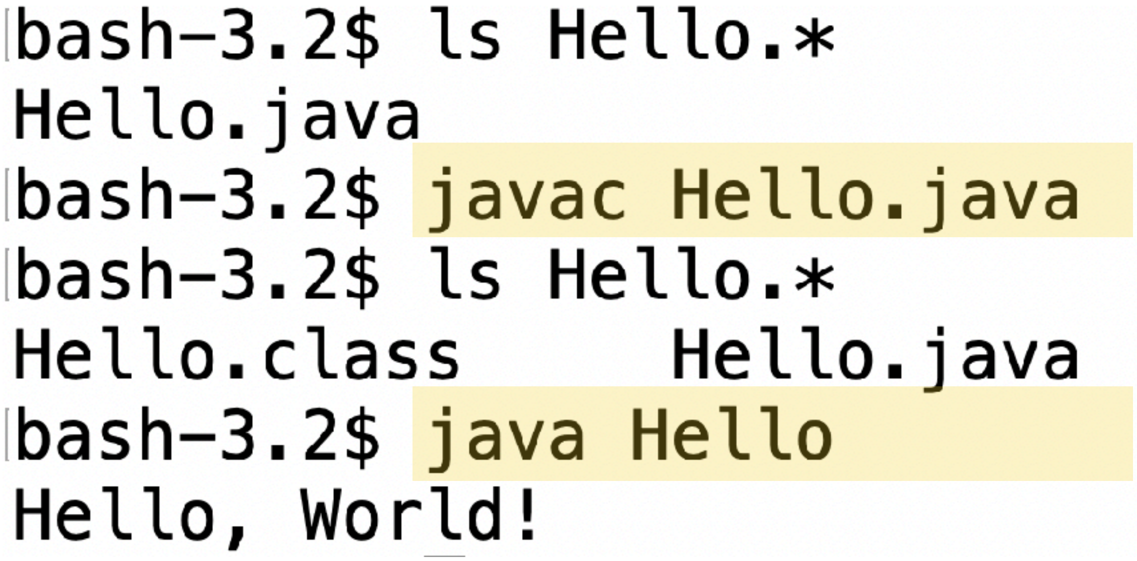
*Technically Java is both interpreted and compiled, but we can ignore that detail for now.

Using the Java Compiler

- The compiler converts our Java source code into compiled byte code which is faster to run (hence the performance benefits)
- Java source files are always named
  `<file>.java`
- To compile, type:
  `javac <file>.java`
- Compilers detect and report syntax errors before execution
- Compiler creates class files:
  `<file>.class`
- Code is executed by typing
  `java <file>`
  (without the .class extension)



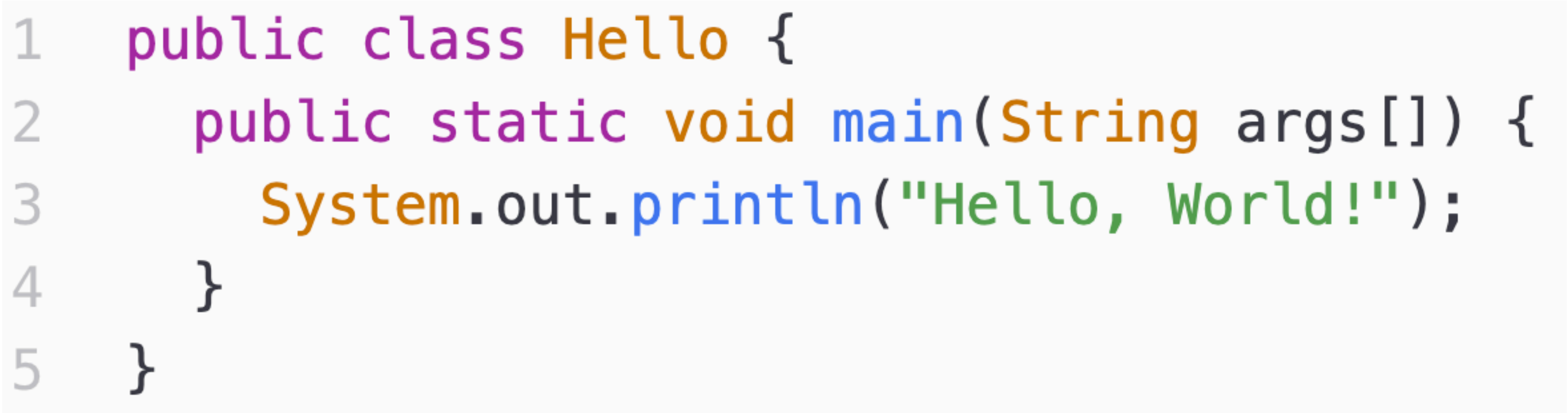
```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```



```
bash-3.2$ ls Hello.*
Hello.java
bash-3.2$ javac Hello.java
bash-3.2$ ls Hello.*
Hello.class     Hello.java
bash-3.2$ java Hello
Hello, World!
```

Important Java Rules

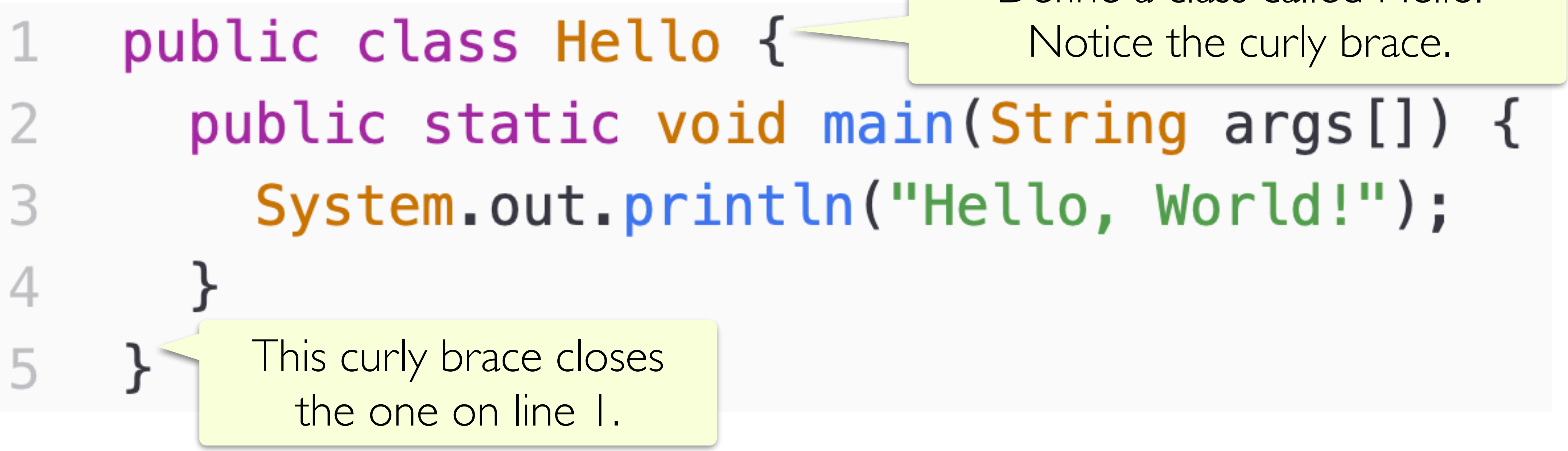
- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (`Hello.java`).
- Every object in Java must have an **explicit type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code contained within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)



```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

Important Java Rules

- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (`Hello.java`).
- Every object in Java must have an **explicit type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code contained within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)



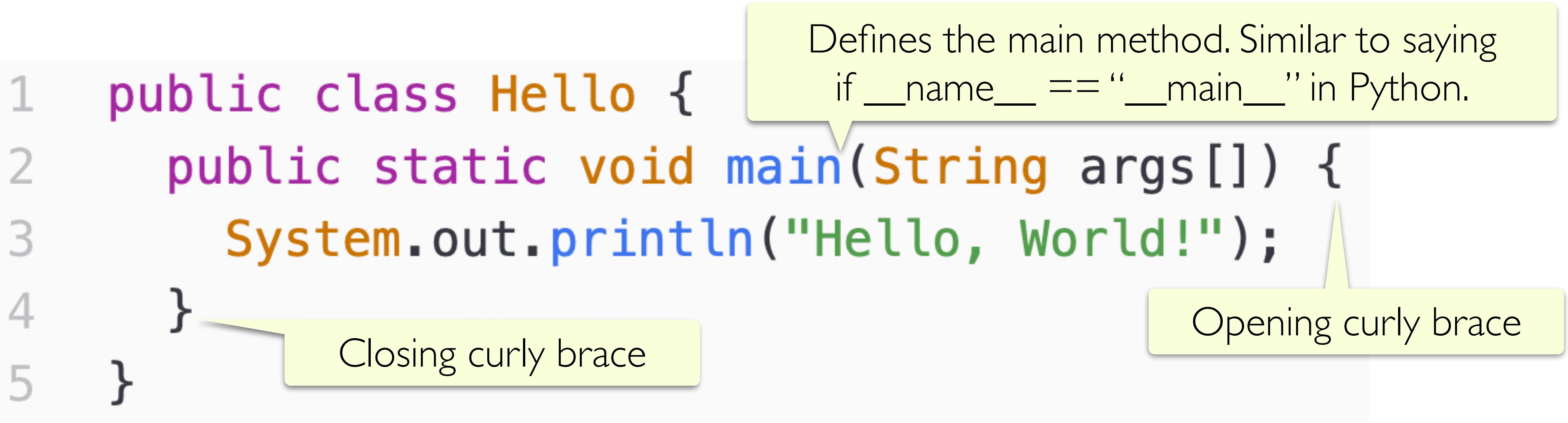
```
1 public class Hello {
2   public static void main(String args[]) {
3     System.out.println("Hello, World!");
4   }
5 }
```

Define a class called Hello. Notice the curly brace.

This curly brace closes the one on line 1.

Important Java Rules

- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (`Hello.java`).
- Every object in Java must have an **explicit type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code contained within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)



```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

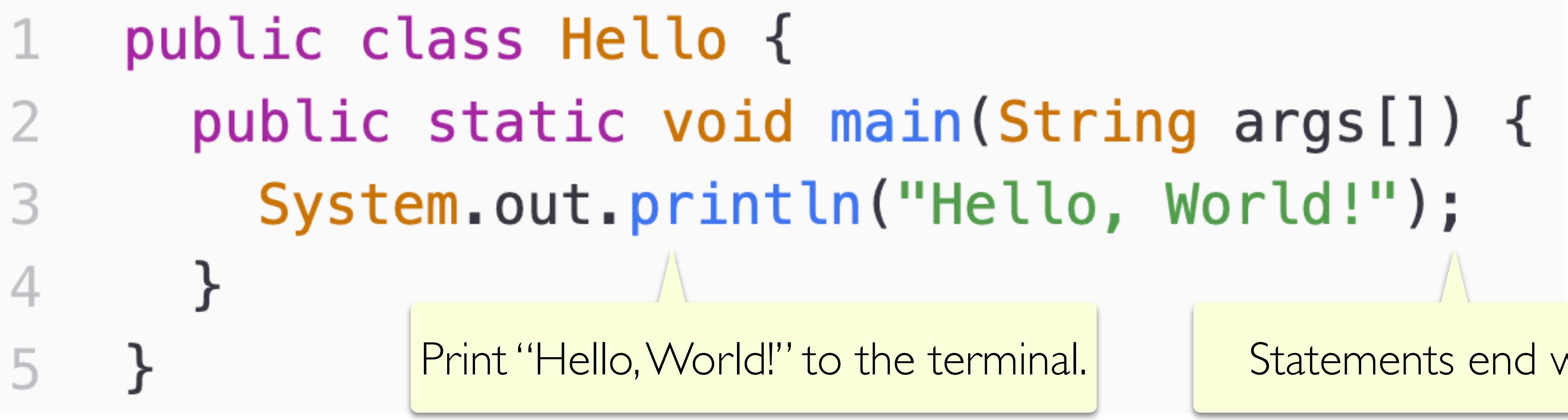
Defines the main method. Similar to saying if __name__ == "__main__" in Python.

Opening curly brace

Closing curly brace

Important Java Rules

- Every Java program must define a **class**, and all code is inside a class.
- The **file name** must be the same as the class name (`Hello.java`).
- Every object in Java must have an **explicit type**.
- Every Java program that we want to execute must have a main method: `public static void main(String[] args)`
- Blocks of code contained within `{}` (versus indentation in Python)
- Statements end with `;` (versus new line in Python)

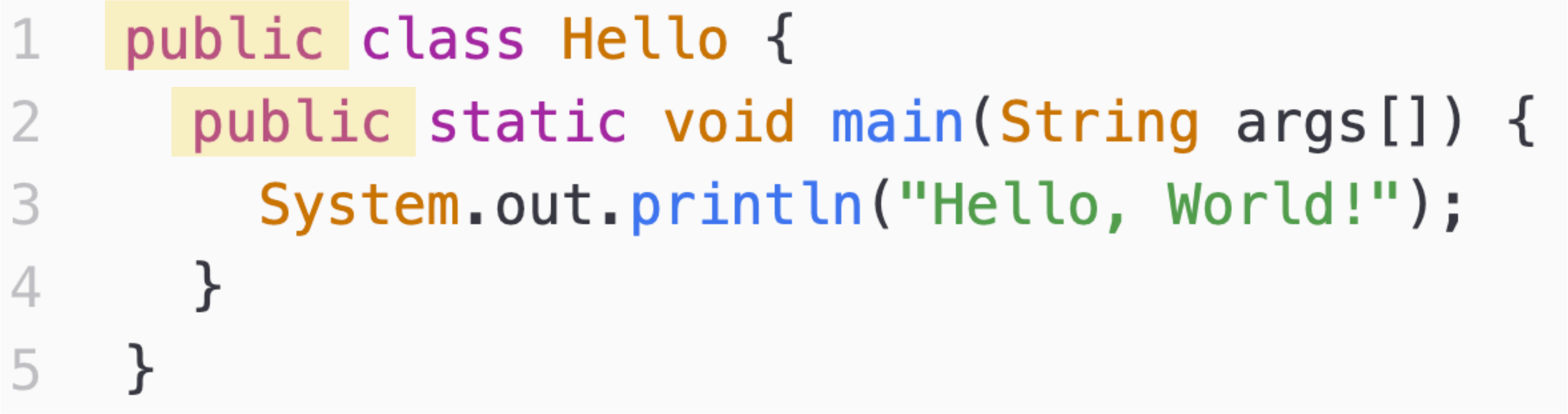


```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

Print "Hello, World!" to the terminal.

Statements end with a ;

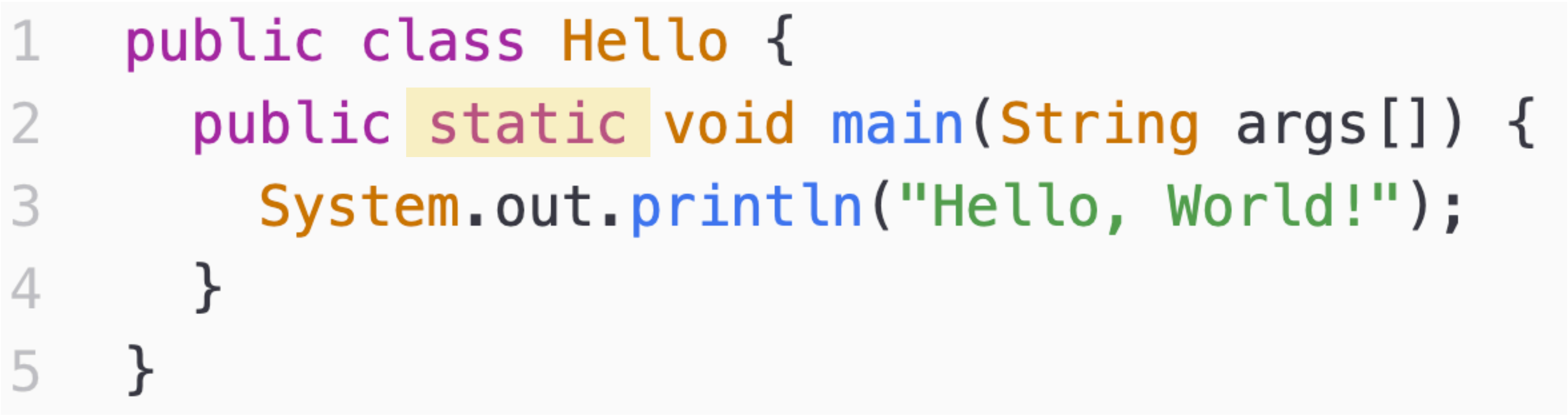
Public, Private, Protected



```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- `public` indicates to the Java compiler that this is a method that anyone can call
- Java enforces several levels of security on methods (also variables and classes): `public`, `protected`, and `private`
- Similar to _ and __ methods in Python, but more strictly enforced

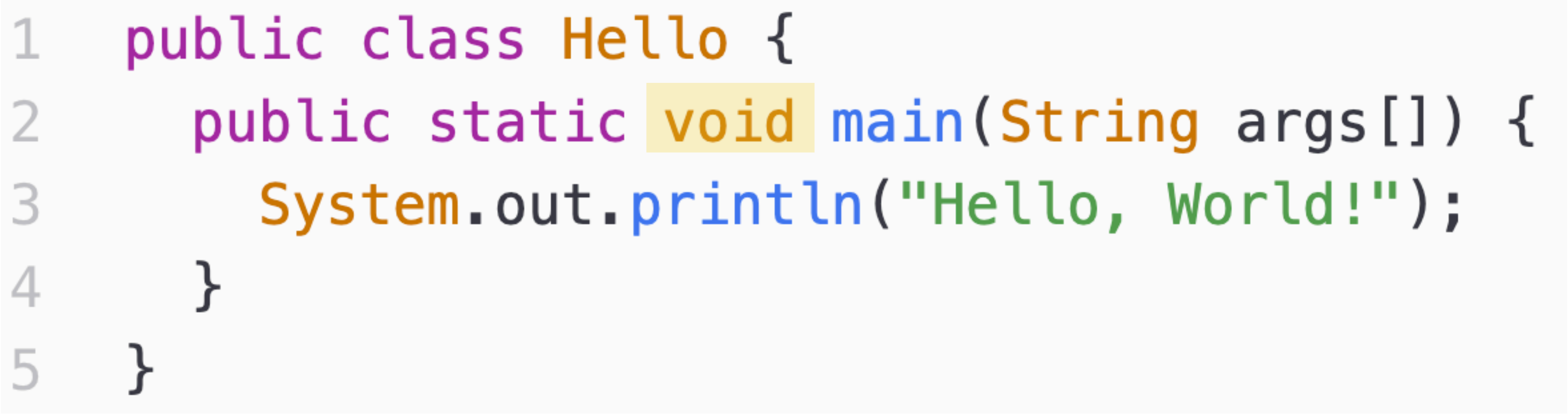
static



```
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- `static` indicates that this is a method that is part of the class, but is not a method for any ***one instance*** of the class (`static` exists in both Java and Python!)
- Most methods we used in Python required an instance of the class in order for the method to be called:
  - Example: `s.upper()` (where `s` is a string and `upper()` is a method in the string class)
- With a static method, the object to the left of the . is a class, ***not an instance*** of the class.
- For example the way that we would call the main method directly is: `Hello.main(…)`.
- Similar to Python modules (such as random) that don't require an instance
  - Example: `random.randint(0,15)`

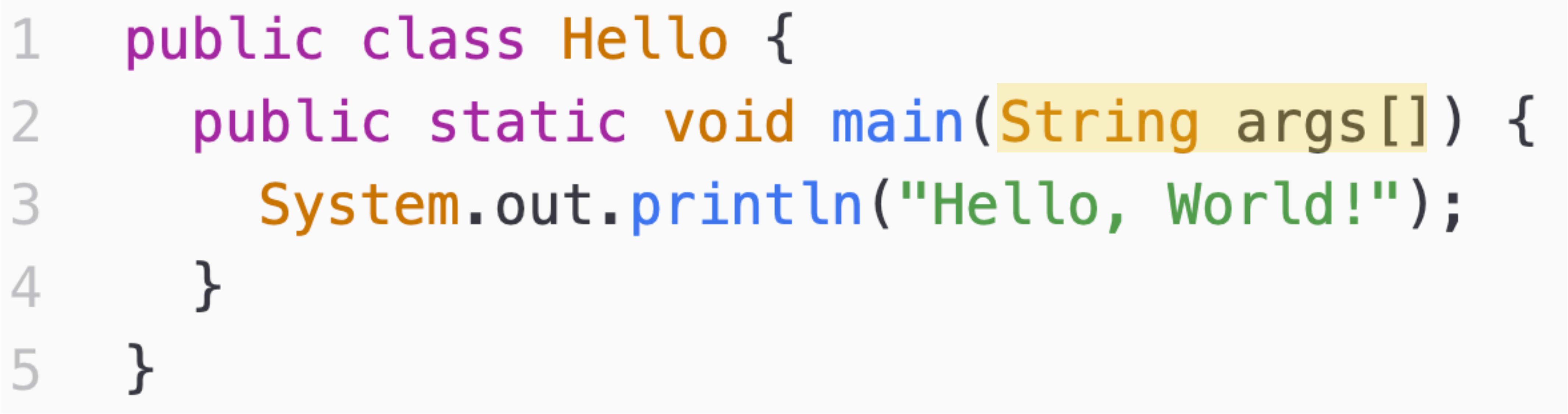
void



```java
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- `void` tells the Java compiler that this method ***will not return a value***
- `void` means "no type"
- Roughly analogous to omitting the return statement in a Python method (or having an ***implicit*** return of None)

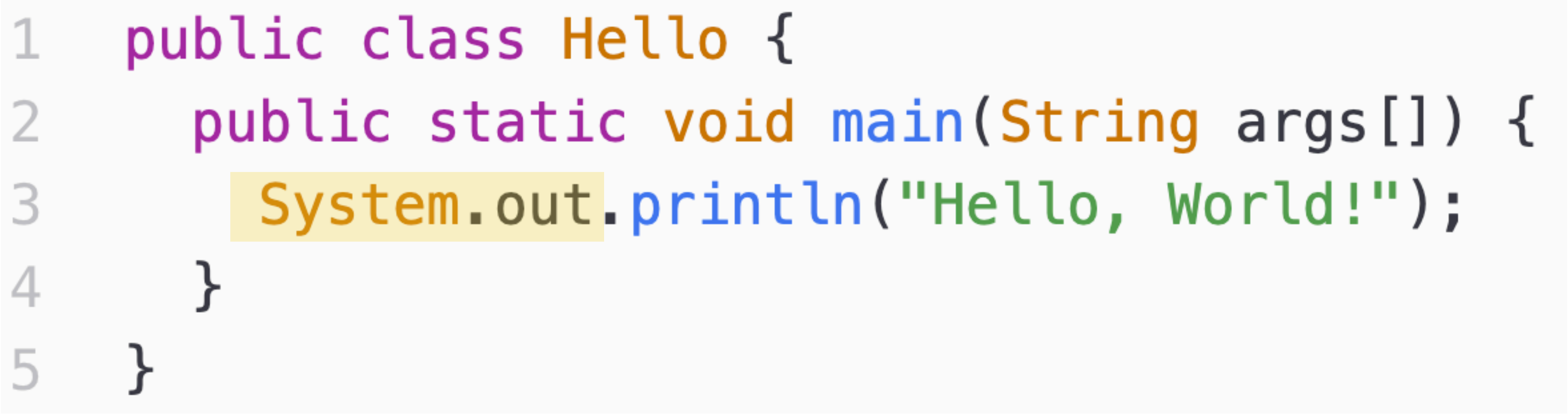
String args[ ]



```
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- Our main method takes as input an **array** (denoted by `[]`) of `Strings` called `args`
  - This is used for handling command-line arguments but we won't worry about that now
- Since everything in Java must have a type, we also have to tell the compiler that the types of values stored in our array are Strings
- Recall that arrays are a lot like lists in Python

System.out and System.in



```
public class Hello {
  public static void main(String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- `System` is a Java class
- Within the System class we find the object named `out`
- The `out` object is the ***standard output stream*** for this program. The `in` object is the ***standard input stream***. We'll come back to that soon.
- The `println` method prints a string with a newline character at the end
- Anywhere in Python that you used the `print(…)` function you will use the `System.out.println(…)` method in Java

Moving on…

Programming Language Features

- **Basic features:**
  - Data Types
  - Reading user input
  - Loops
  - Conditionals
- **Advanced topics:**
  - Classes
  - Interfaces
  - Collections
  - Graphical User Interface Programming

We have extensively studied all of these features in Python. Let's compare and contrast with Java.

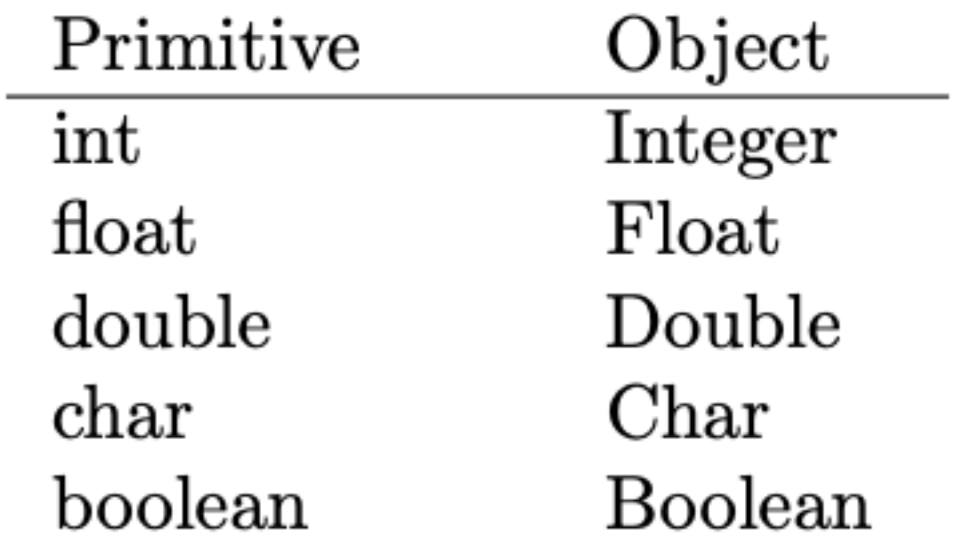
Programming Language Features

- **Basic features:**
  - Data Types
  - Reading user input
  - Loops
  - Conditionals
- **Advanced topics:**
  - Classes
  - Interfaces
  - Collections
  - Graphical User Interface Programming

Let's start with data types and reading user input.

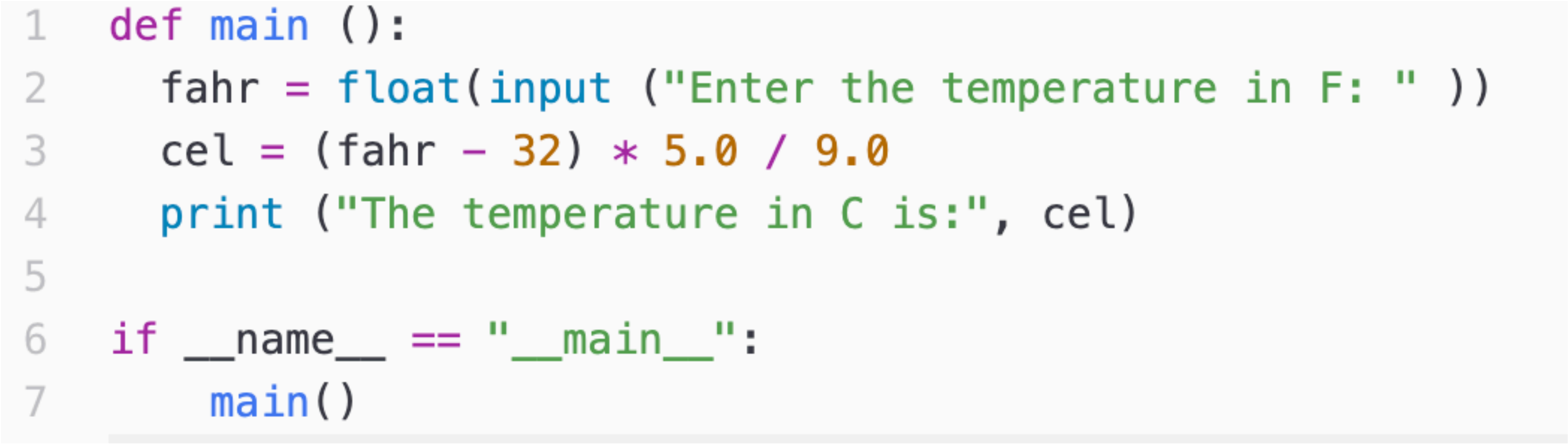
Basic Data Types

- All **data types** in Python are **objects**
  - Implemented using **classes** and **methods** just like our LinkedList
- Two types of data types in Java: **primitive** (non-objects) and **Objects**
  - Example: `int` (lowercase) and `Integer` (uppercase)
  - The benefit of primitive data types is fast operations
  - We'll mostly use the Object versions and let the compiler handle conversions to primitives for us
- Java data types:



Primitive	Object
int	Integer
float	Float
double	Double
char	Char
boolean	Boolean

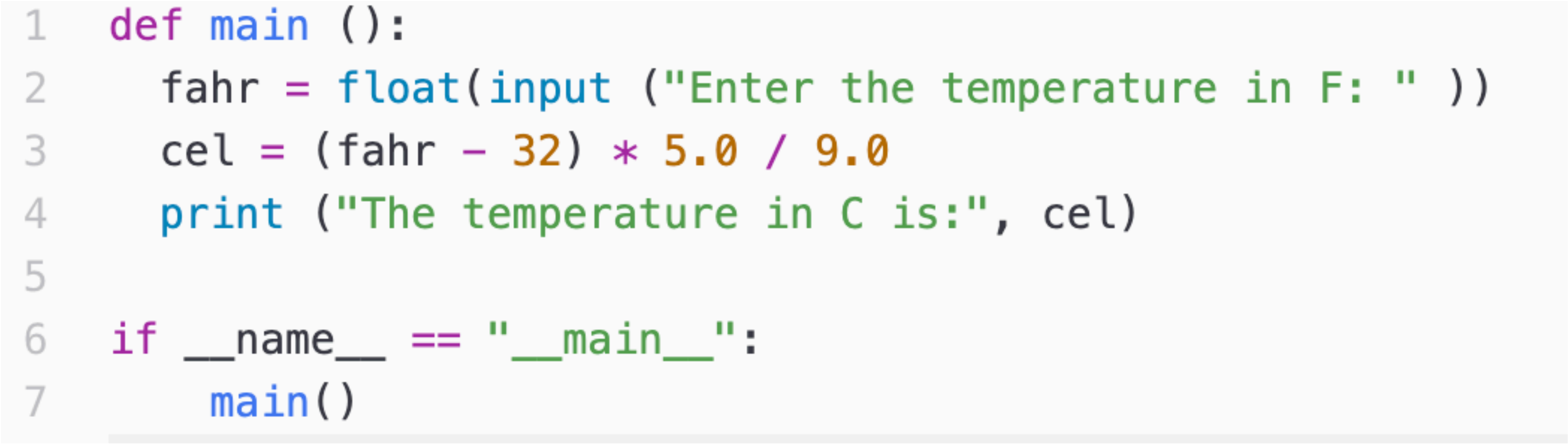
Simple Example



```
def main ():
  fahr = float(input ("Enter the temperature in F: " ))
  cel = (fahr - 32) * 5.0 / 9.0
  print ("The temperature in C is:", cel)

if __name__ == "__main__":
    main()
```

- Consider this Python script: `temp.py`
- What does it do?

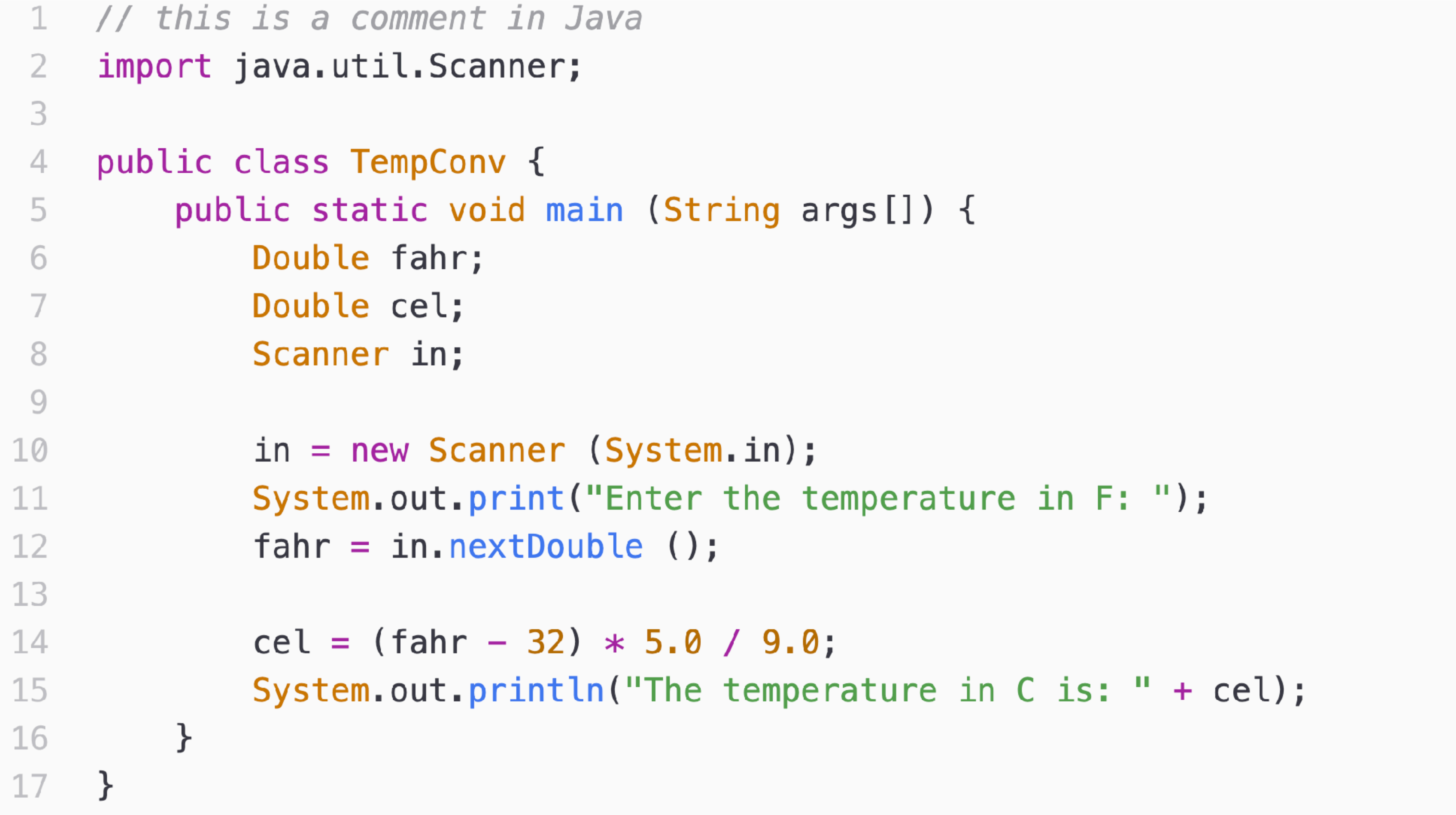
Simple Example



```
def main ():
  fahr = float(input ("Enter the temperature in F: " ))
  cel = (fahr - 32) * 5.0 / 9.0
  print ("The temperature in C is:", cel)

if __name__ == "__main__":
    main()
```

- Consider this Python script: `temp.py`
- What does it do?
    - Asks user to enter a temperature in Fahrenheit and converts the string input to float
    - Does the computation to convert temperature to Celsius
    - Prints result

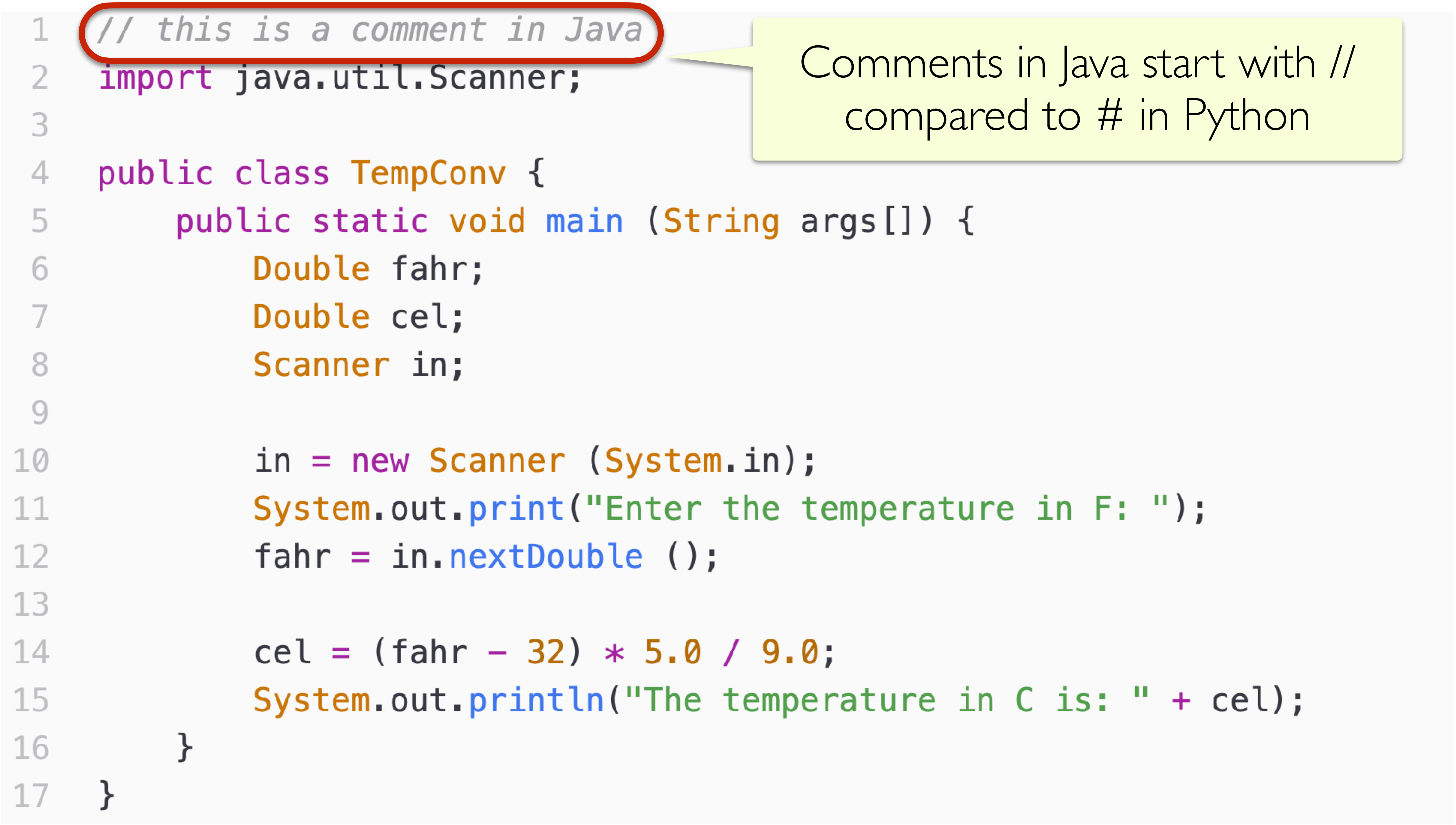
Simple Example



```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

- Same program in Java: `TempConv.java`

Simple Example

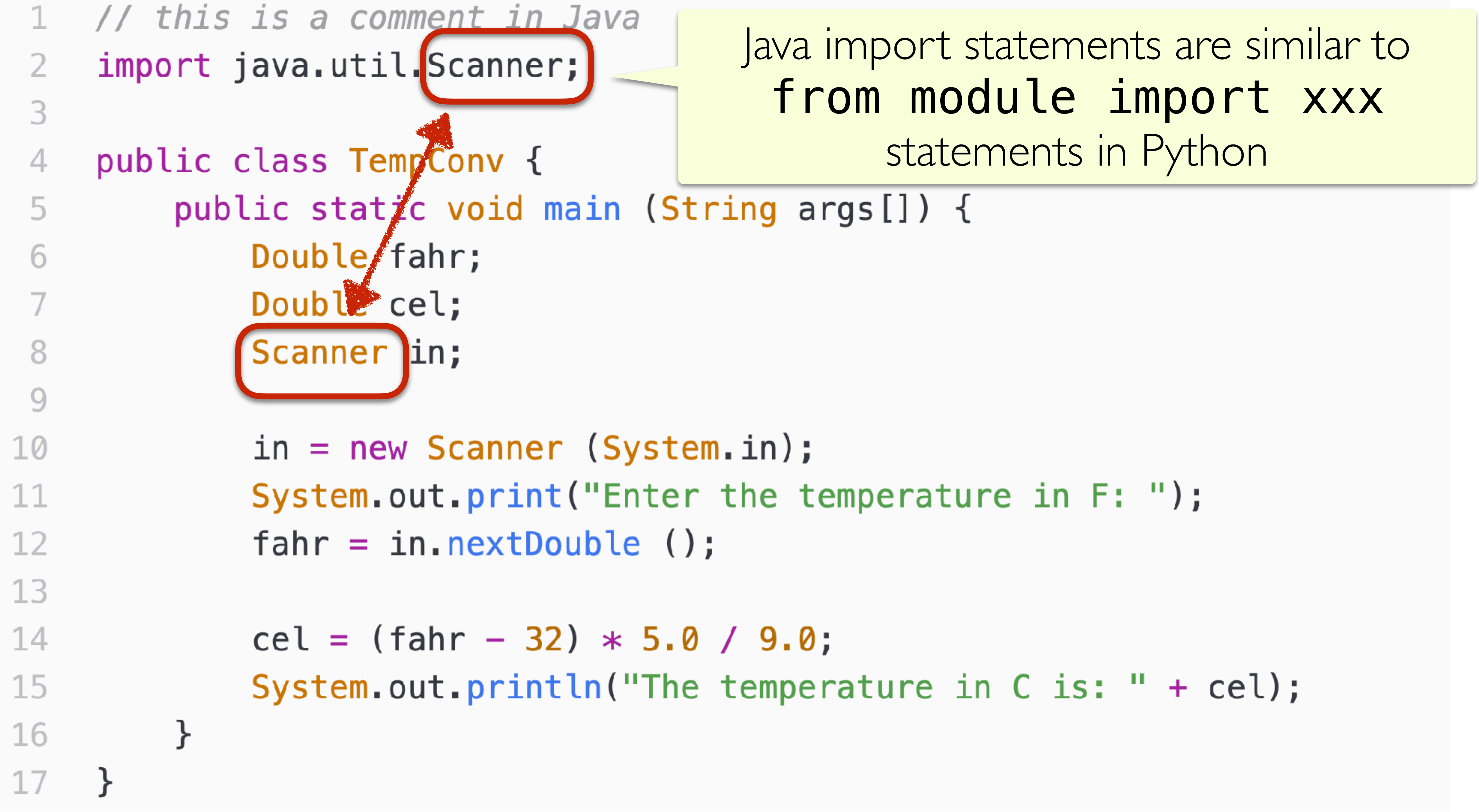


```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

Comments in Java start with // compared to # in Python

- Same program in Java: `TempConv.java`

Simple Example

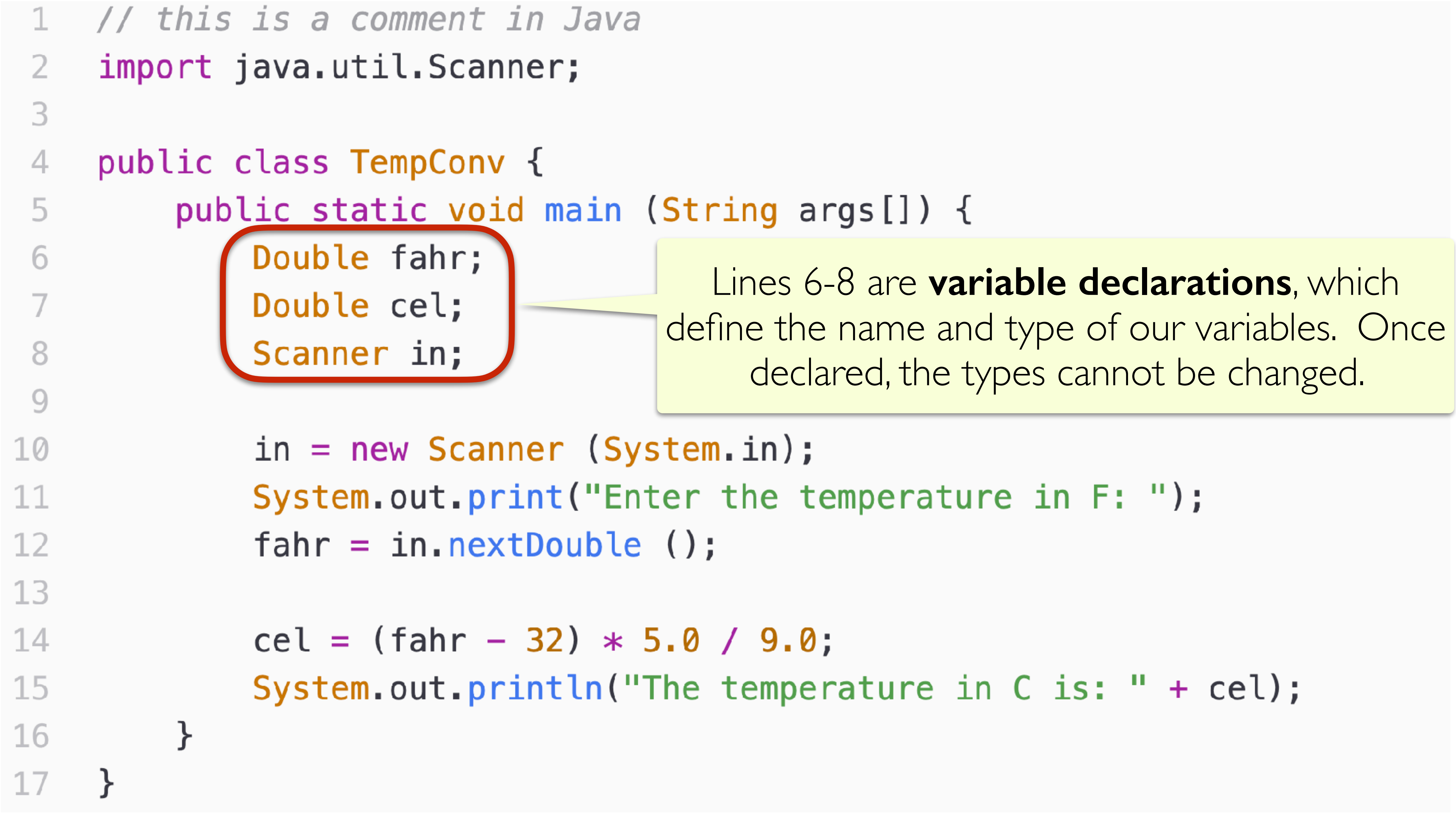


```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

Java import statements are similar to
`from module import xxx`
statements in Python

- Java uses import statements to tell the compiler what classes to use

Simple Example

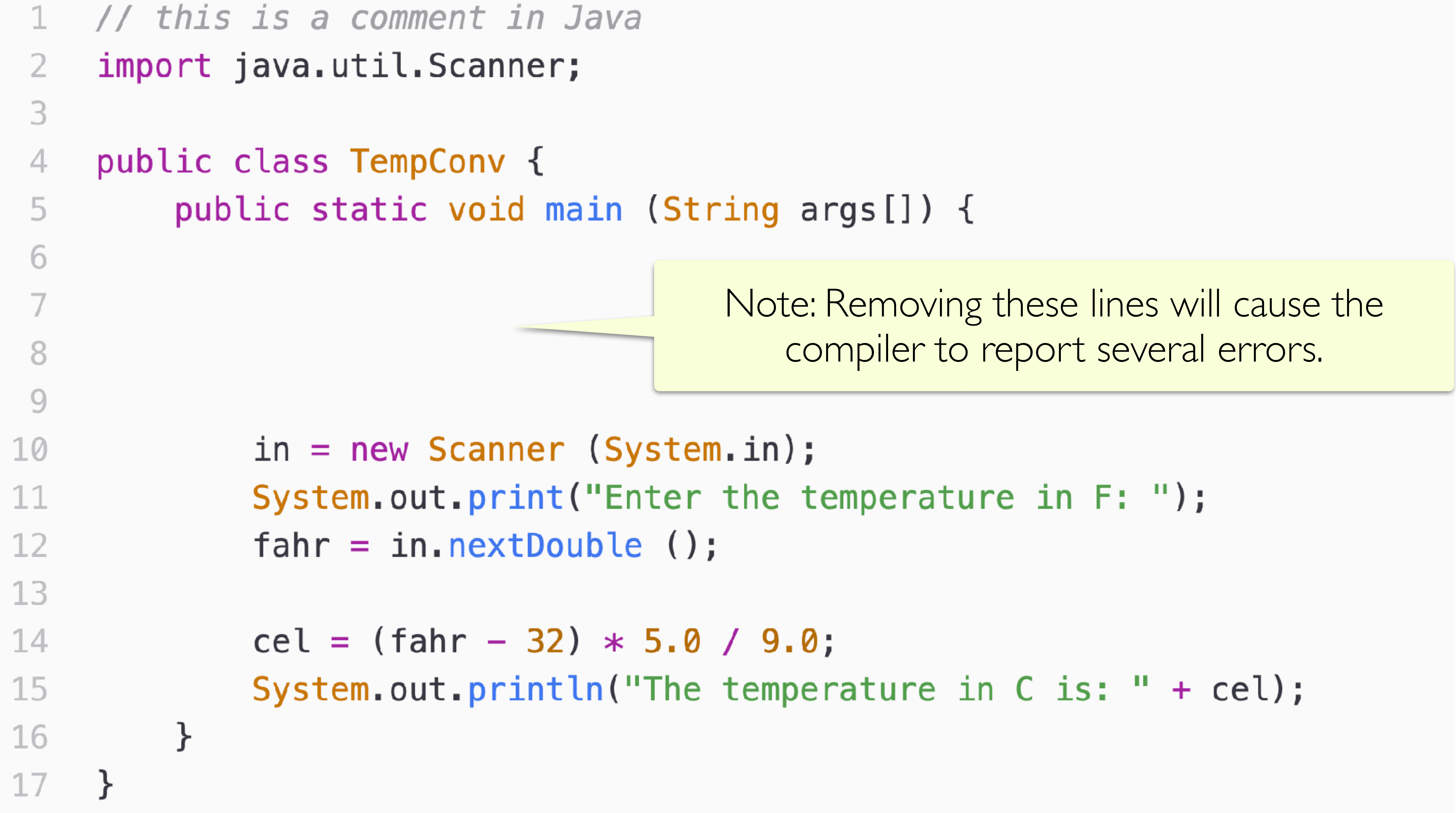


```
1  // this is a comment in Java
2  import java.util.Scanner;
3
4  public class TempConv {
5      public static void main (String args[]) {
6          Double fahr;
7          Double cel;
8          Scanner in;
9
10         in = new Scanner (System.in);
11         System.out.print("Enter the temperature in F: ");
12         fahr = in.nextDouble ();
13
14         cel = (fahr - 32) * 5.0 / 9.0;
15         System.out.println("The temperature in C is: " + cel);
16     }
17 }
```

Lines 6-8 are **variable declarations**, which define the name and type of our variables. Once declared, the types cannot be changed.

- Java is **statically typed**. Thus, all variables must be **declared** with a name and type before they are used. Common convention is to declare variables at the top of our methods/classes.

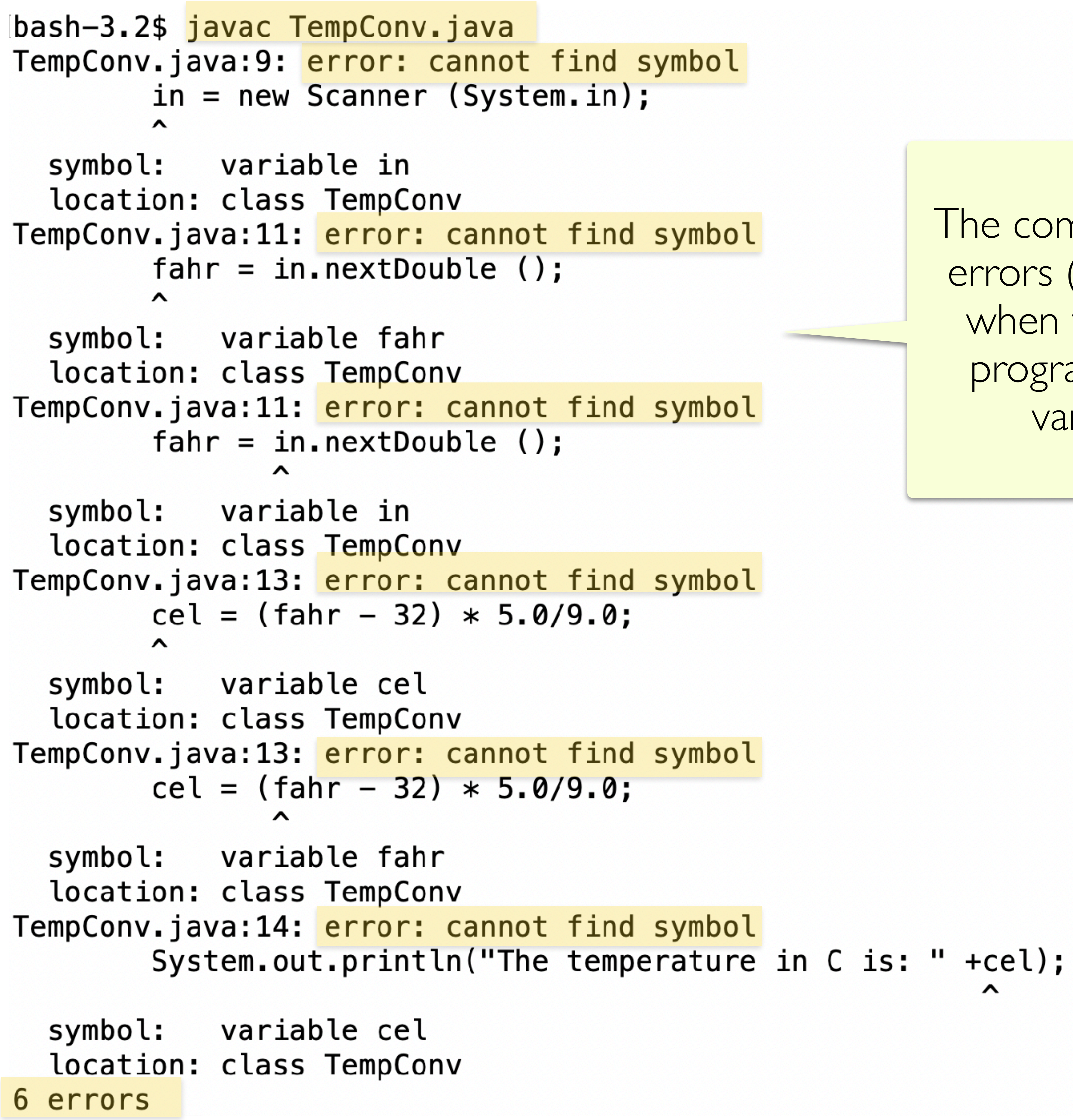
Simple Example



```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {




        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

Note: Removing these lines will cause the compiler to report several errors.

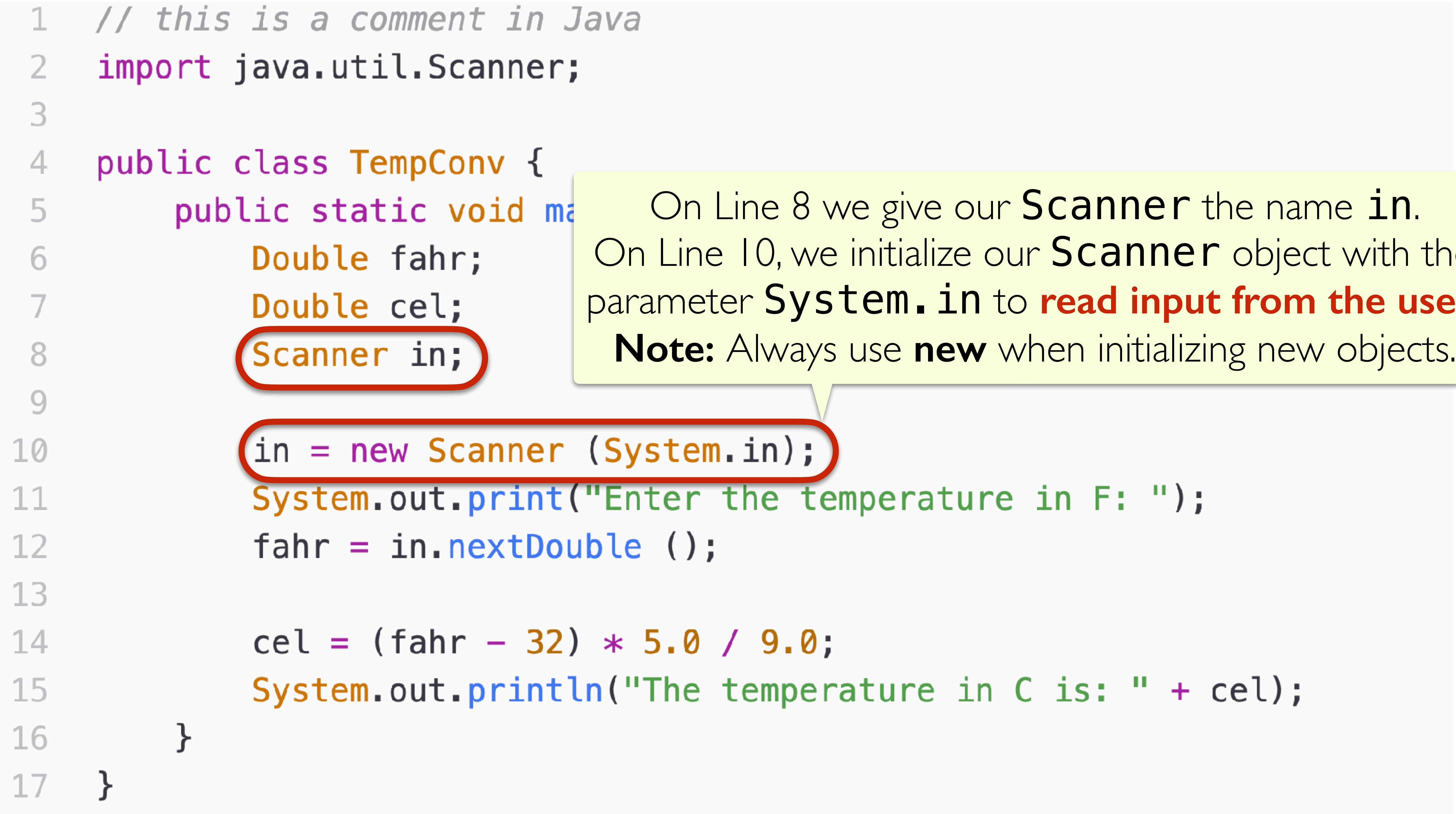
- Let’s try to compile: `javac TempConv.java`



```
bash-3.2$ javac TempConv.java
TempConv.java:9: error: cannot find symbol
        in = new Scanner (System.in);
        ^

  symbol:   variable in
  location: class TempConv
TempConv.java:11: error: cannot find symbol
        fahr = in.nextDouble ();
        ^

  symbol:   variable fahr
  location: class TempConv
TempConv.java:11: error: cannot find symbol
        fahr = in.nextDouble ();
               ^

  symbol:   variable in
  location: class TempConv
TempConv.java:13: error: cannot find symbol
        cel = (fahr - 32) * 5.0/9.0;
        ^

  symbol:   variable cel
  location: class TempConv
TempConv.java:13: error: cannot find symbol
        cel = (fahr - 32) * 5.0/9.0;
               ^

  symbol:   variable fahr
  location: class TempConv
TempConv.java:14: error: cannot find symbol
        System.out.println("The temperature in C is: " +cel);
                                                        ^

  symbol:   variable cel
  location: class TempConv
6 errors
```

The compiler will report several errors (sometimes repeatedly) when we try to compile our program after removing our variable declarations.

Simple Example

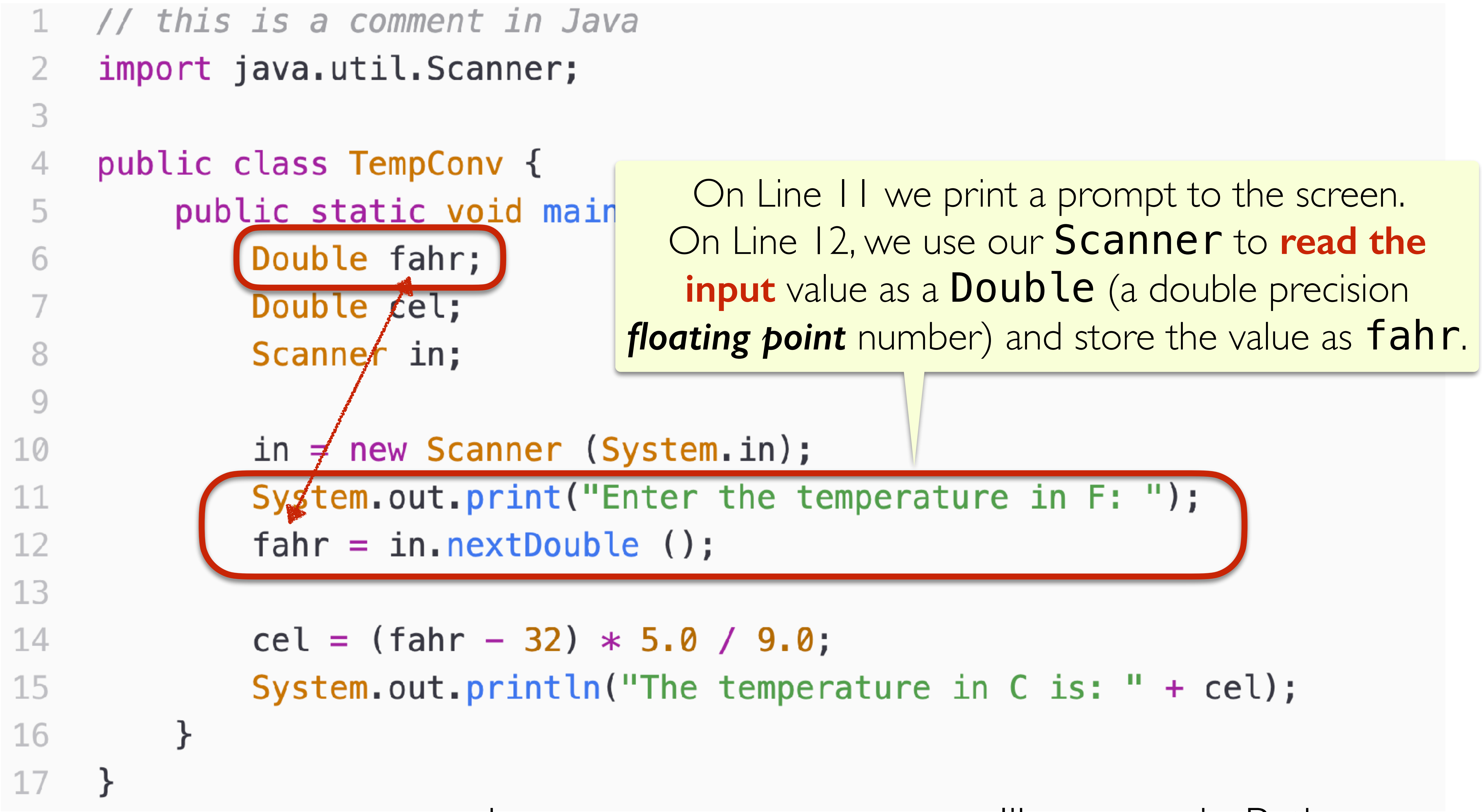


```
1   // this is a comment in Java
2   import java.util.Scanner;
3
4   public class TempConv {
5       public static void ma
6           Double fahr;
7           Double cel;
8           Scanner in;
9
10          in = new Scanner (System.in);
11          System.out.print("Enter the temperature in F: ");
12          fahr = in.nextDouble ();
13
14          cel = (fahr - 32) * 5.0 / 9.0;
15          System.out.println("The temperature in C is: " + cel);
16      }
17  }
```

On Line 8 we give our `Scanner` the name `in`.
On Line 10, we initialize our `Scanner` object with the parameter `System.in` to **read input from the user.**
Note: Always use **new** when initializing new objects.

- After declaring a `Scanner` object named `in`, we also have to initialize it before using it (like calling `__init__()` in Python).

Simple Example



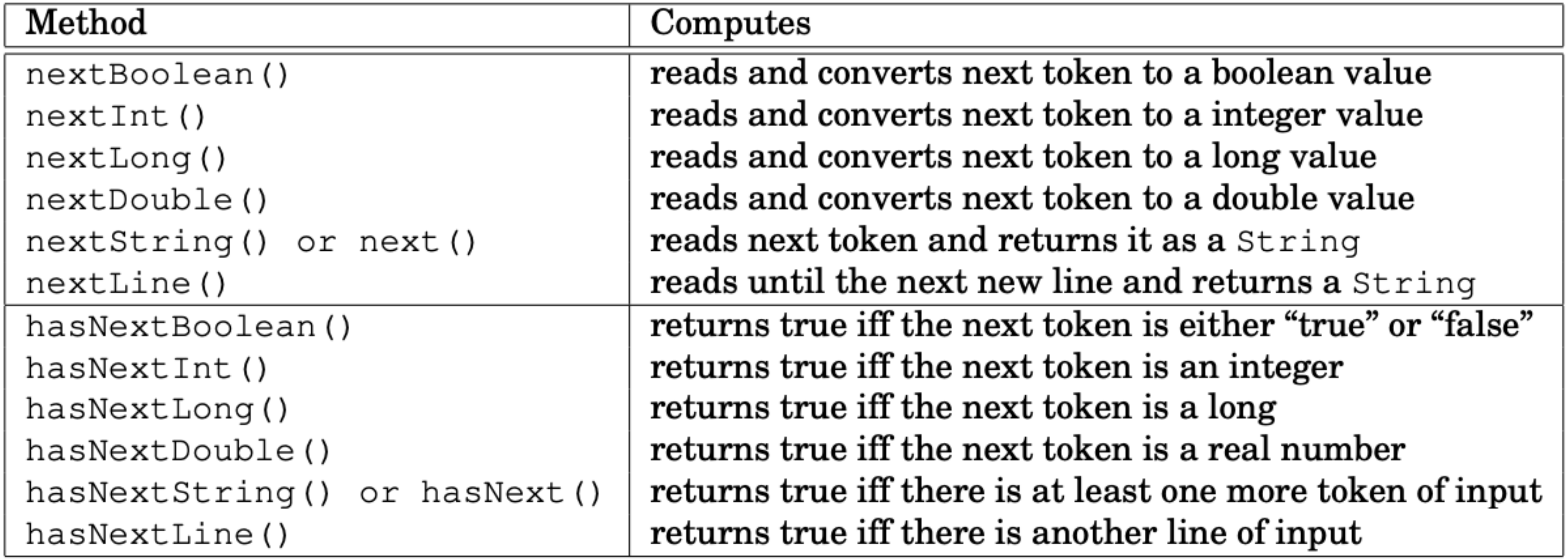
```
1   // this is a comment in Java
2   import java.util.Scanner;
3
4   public class TempConv {
5       public static void main
6           Double fahr;
7           Double cel;
8           Scanner in;
9
10          in = new Scanner (System.in);
11          System.out.print("Enter the temperature in F: ");
12          fahr = in.nextDouble ();
13
14          cel = (fahr - 32) * 5.0 / 9.0;
15          System.out.println("The temperature in C is: " + cel);
16      }
17  }
```

On Line 11 we print a prompt to the screen. On Line 12, we use our `Scanner` to **read the input** value as a `Double` (a double precision ***floating point*** number) and store the value as `fahr`.

- `System.out.print` and `System.out.println` are like `print` in Python.
- `in.nextDouble()` automatically reads the user input as a `Double` (like using `input()` in Python and then converting to `float`)

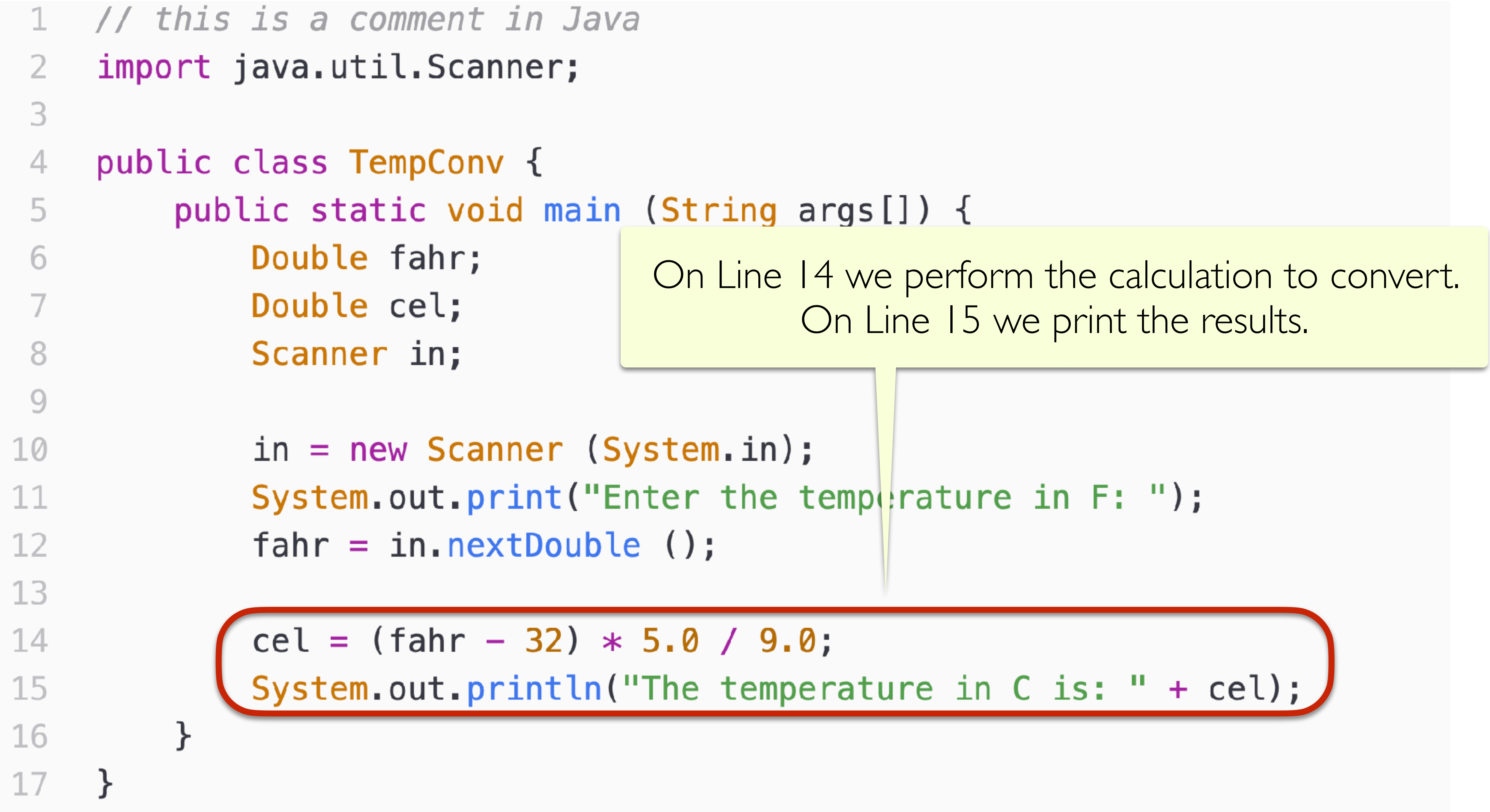
An Aside: Using the Java Scanner

- Since Java is **strongly typed**, we have to be extra careful when reading input from the user to make sure it is of the expected type
- The `Scanner` class provides methods for making sure the next value (like an iterator!) is of the expected type
- Here are some methods for the Java `Scanner` class



Method	Computes
`nextBoolean()`	reads and converts next token to a boolean value
`nextInt()`	reads and converts next token to a integer value
`nextLong()`	reads and converts next token to a long value
`nextDouble()`	reads and converts next token to a double value
`nextString() or next()`	reads next token and returns it as a `String`
`nextLine()`	reads until the next new line and returns a `String`
`hasNextBoolean()`	returns true iff the next token is either "true" or "false"
`hasNextInt()`	returns true iff the next token is an integer
`hasNextLong()`	returns true iff the next token is a long
`hasNextDouble()`	returns true iff the next token is a real number
`hasNextString() or hasNext()`	returns true iff there is at least one more token of input
`hasNextLine()`	returns true iff there is another line of input

Simple Example

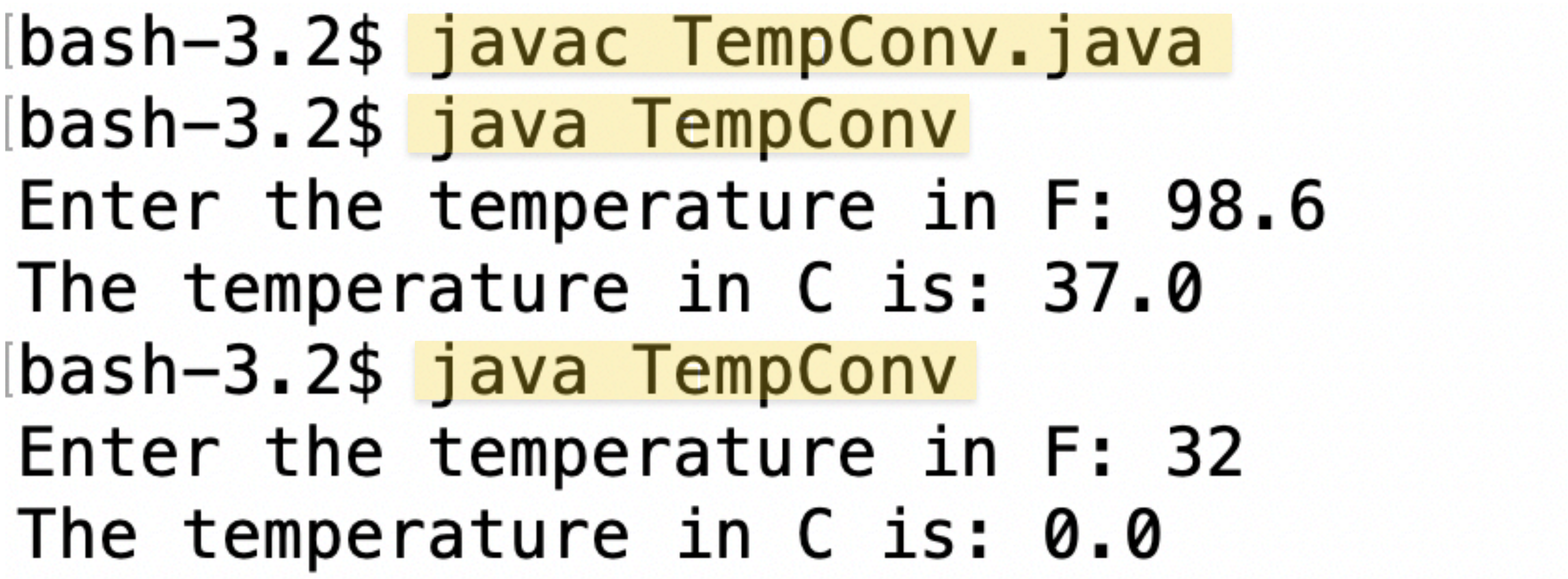


```java
1  // this is a comment in Java
2  import java.util.Scanner;
3
4  public class TempConv {
5      public static void main (String args[]) {
6          Double fahr;
7          Double cel;
8          Scanner in;
9
10         in = new Scanner (System.in);
11         System.out.print("Enter the temperature in F: ");
12         fahr = in.nextDouble ();
13
14         cel = (fahr - 32) * 5.0 / 9.0;
15         System.out.println("The temperature in C is: " + cel);
16     }
17 }
```

On Line 14 we perform the calculation to convert.
On Line 15 we print the results.

- Arithmetic calculations in Java and Python are very similar wrt syntax
- When we print, we use the **+** operator to perform **string concatenation**

Simple Example



```
[bash-3.2$ javac TempConv.java
[bash-3.2$ java TempConv
Enter the temperature in F: 98.6
The temperature in C is: 37.0
[bash-3.2$ java TempConv
Enter the temperature in F: 32
The temperature in C is: 0.0
```

- Before running our program, we compile using `javac`

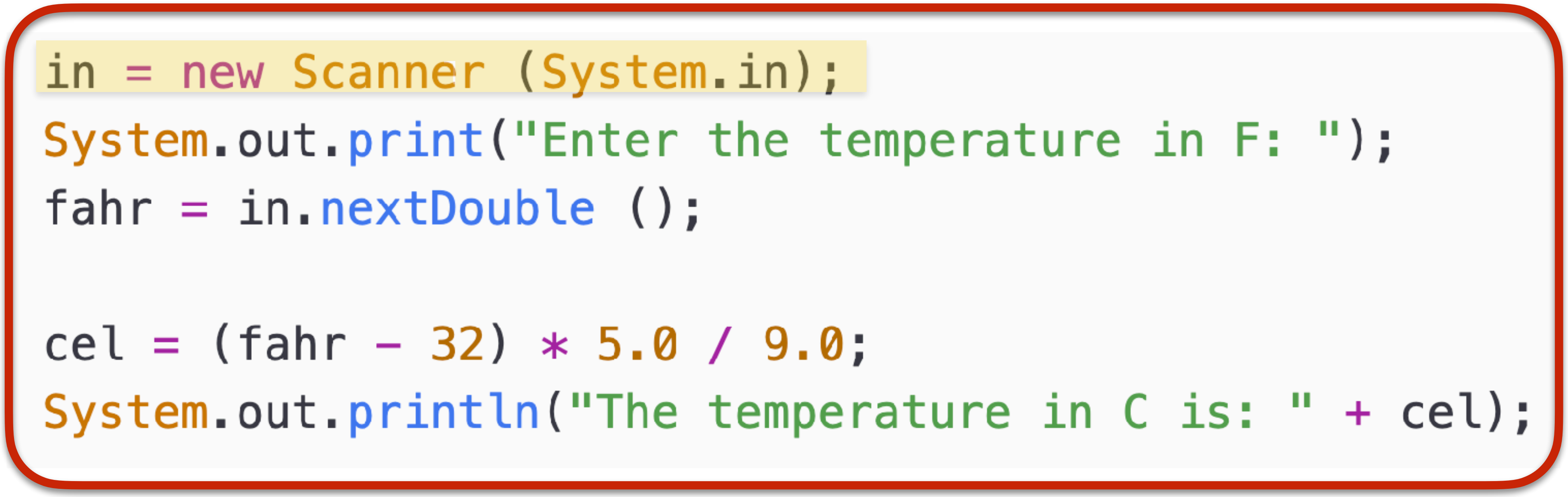
```
javac TempConv.java
```

- To run, we use `java`

```
java TempConv
```

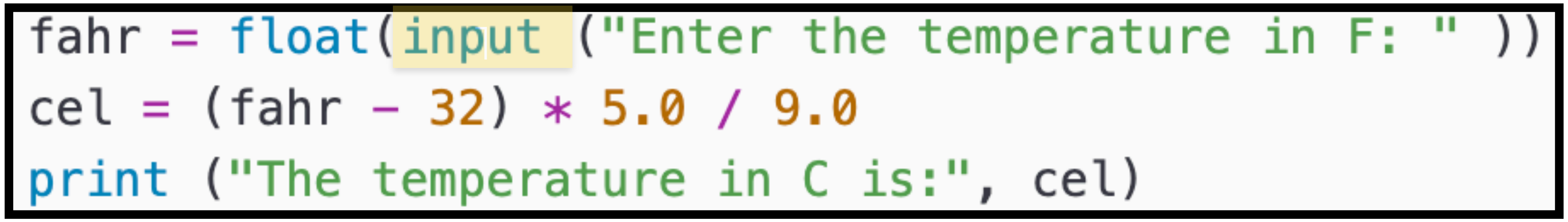
Recap: Python vs. Java

Java:



```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

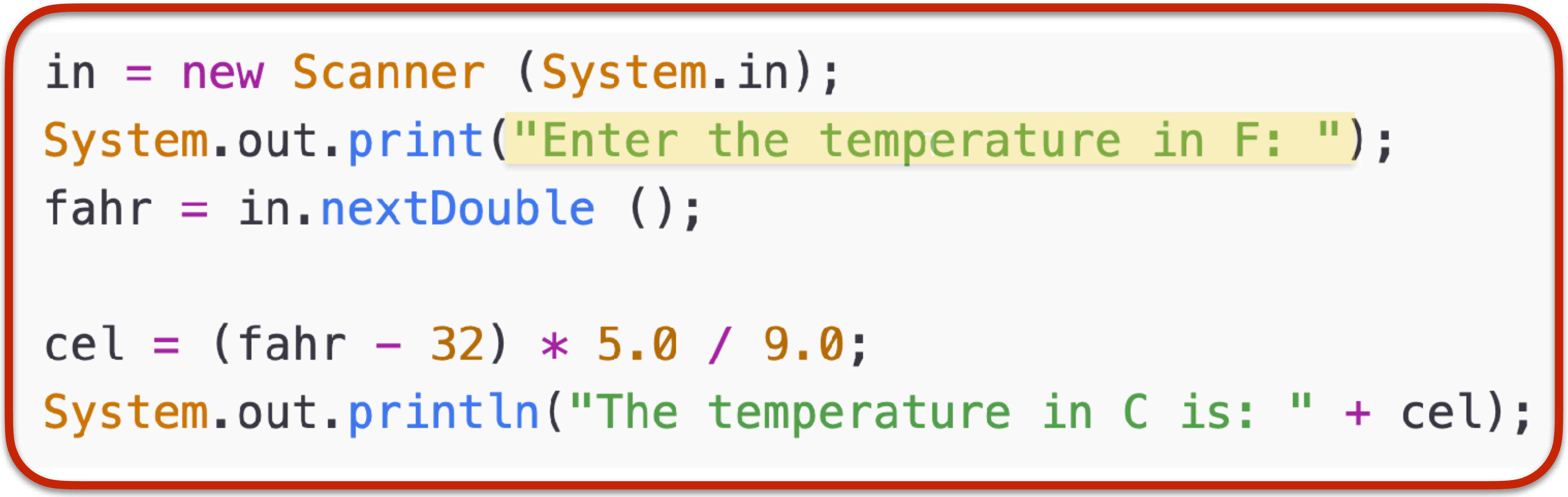


```python
fahr = float(input ("Enter the temperature in F: " ))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 1: Prepare to read input from user.

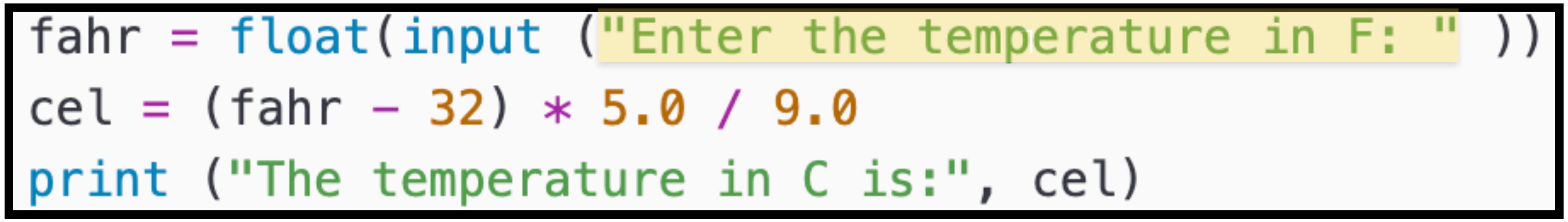
Recap: Python vs. Java

Java:



```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

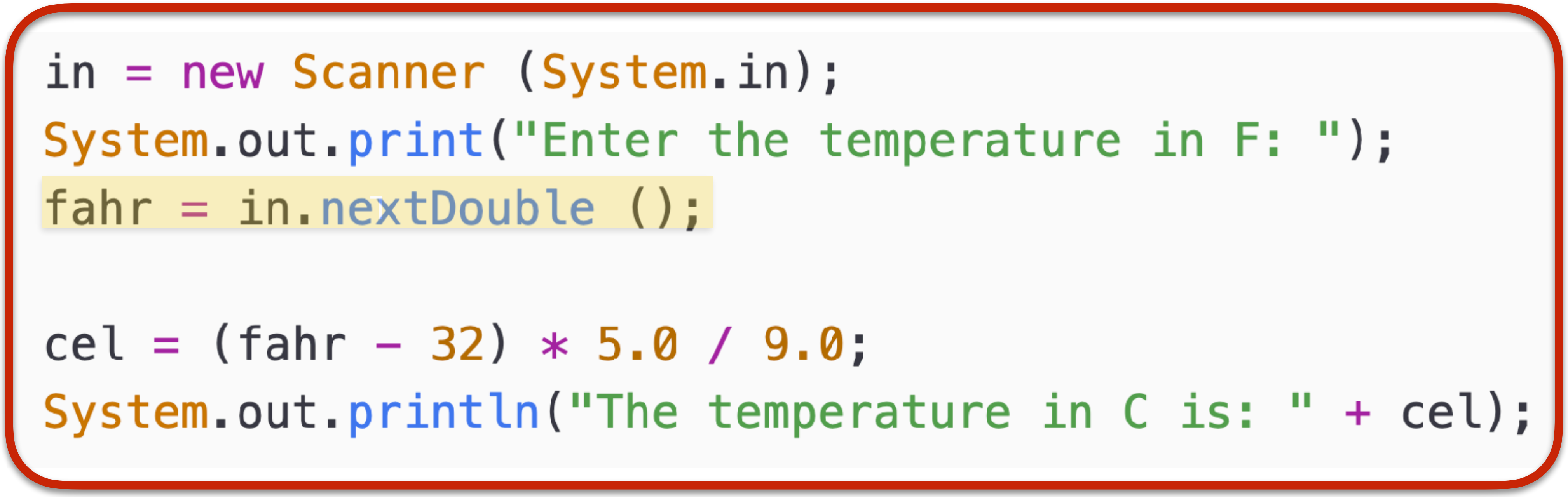


```python
fahr = float(input ("Enter the temperature in F: " ))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 2: Prompt user for input.

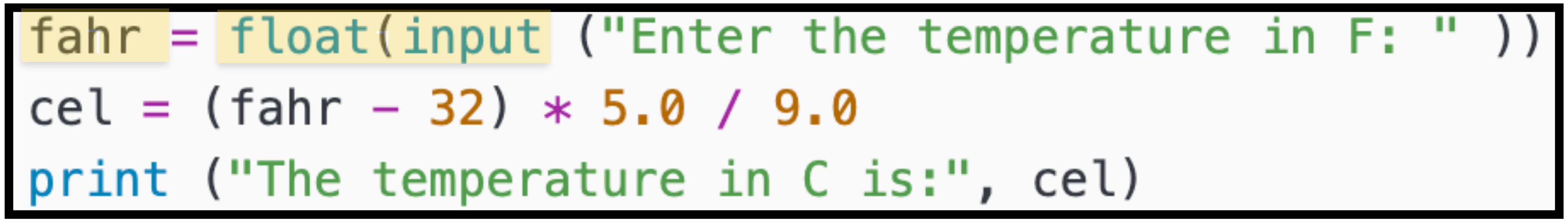
Recap: Python vs. Java

Java:



```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

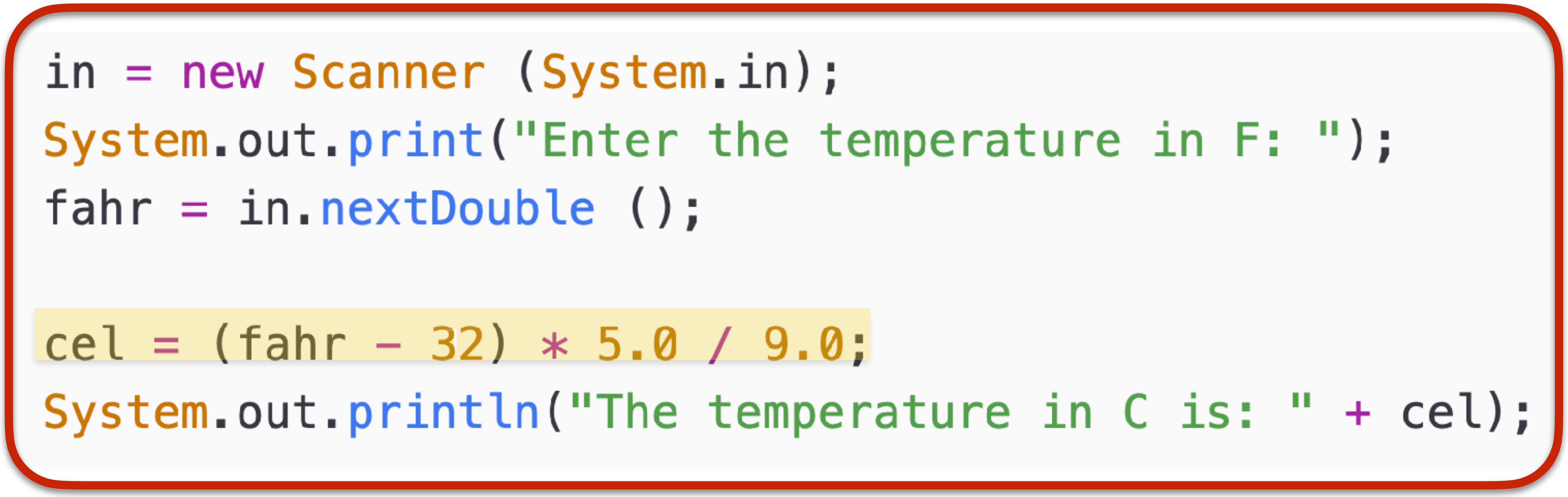


```python
fahr = float(input ("Enter the temperature in F: " ))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 3: Read user input and convert to float/double (that is, a number with a decimal point).

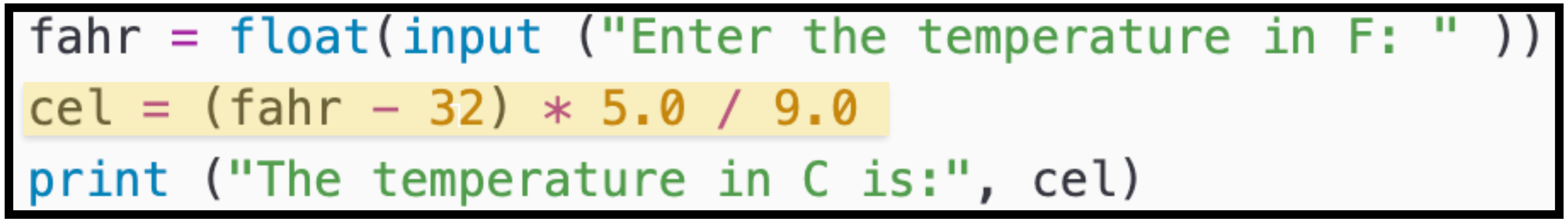
Recap: Python vs. Java

Java:



```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

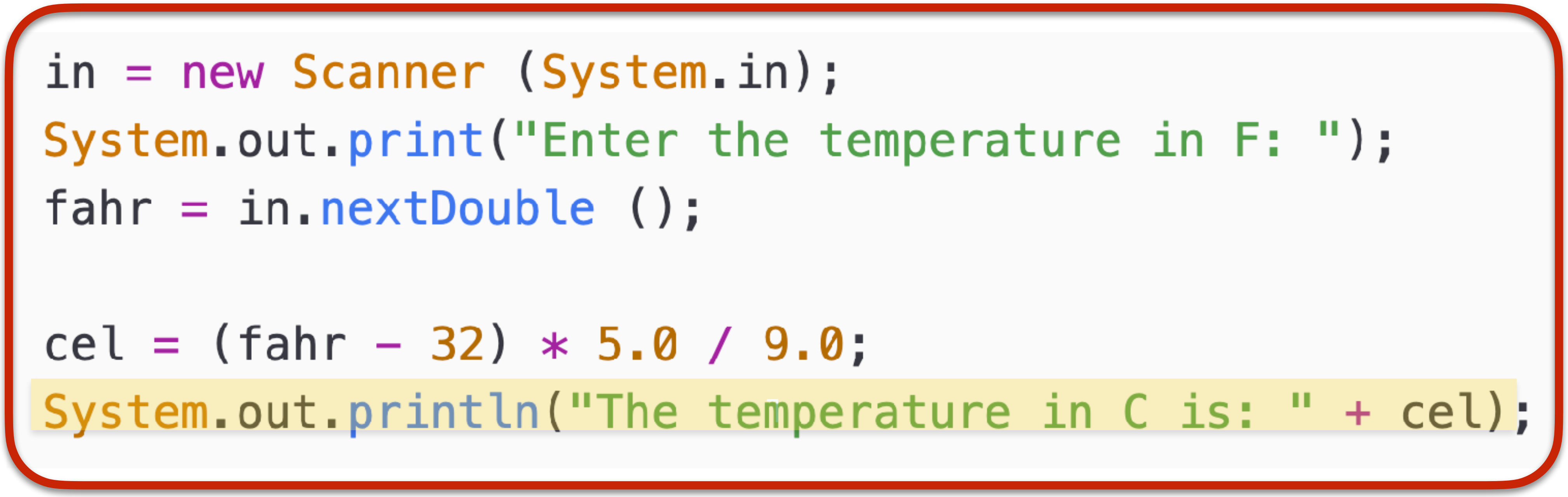


```python
fahr = float(input ("Enter the temperature in F: " ))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 4: Perform conversion to Celsius.

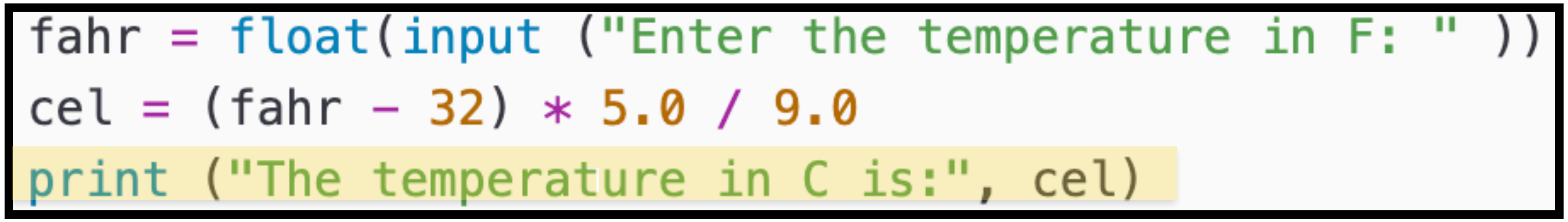
Recap: Python vs. Java

Java:



```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

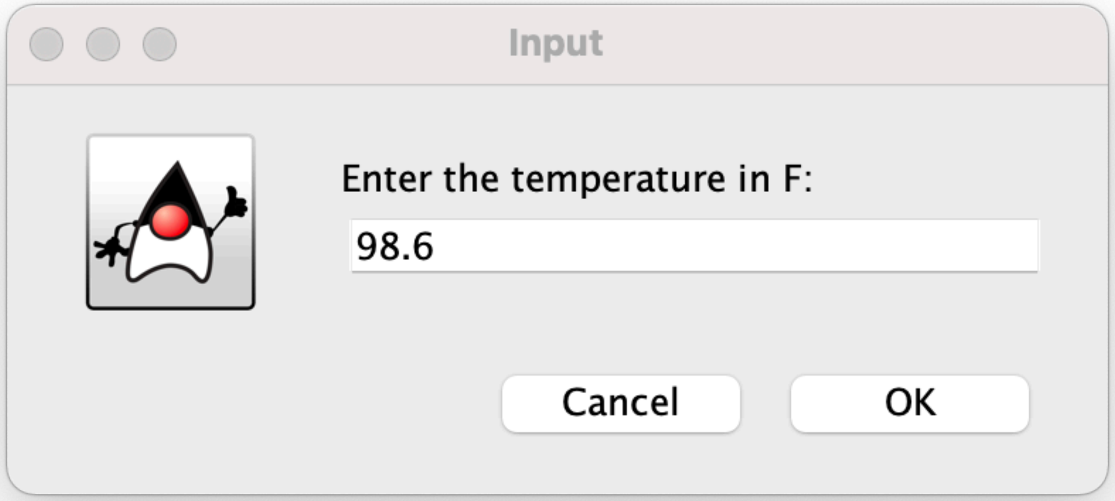


```python
fahr = float(input ("Enter the temperature in F: " ))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 5: Print result.

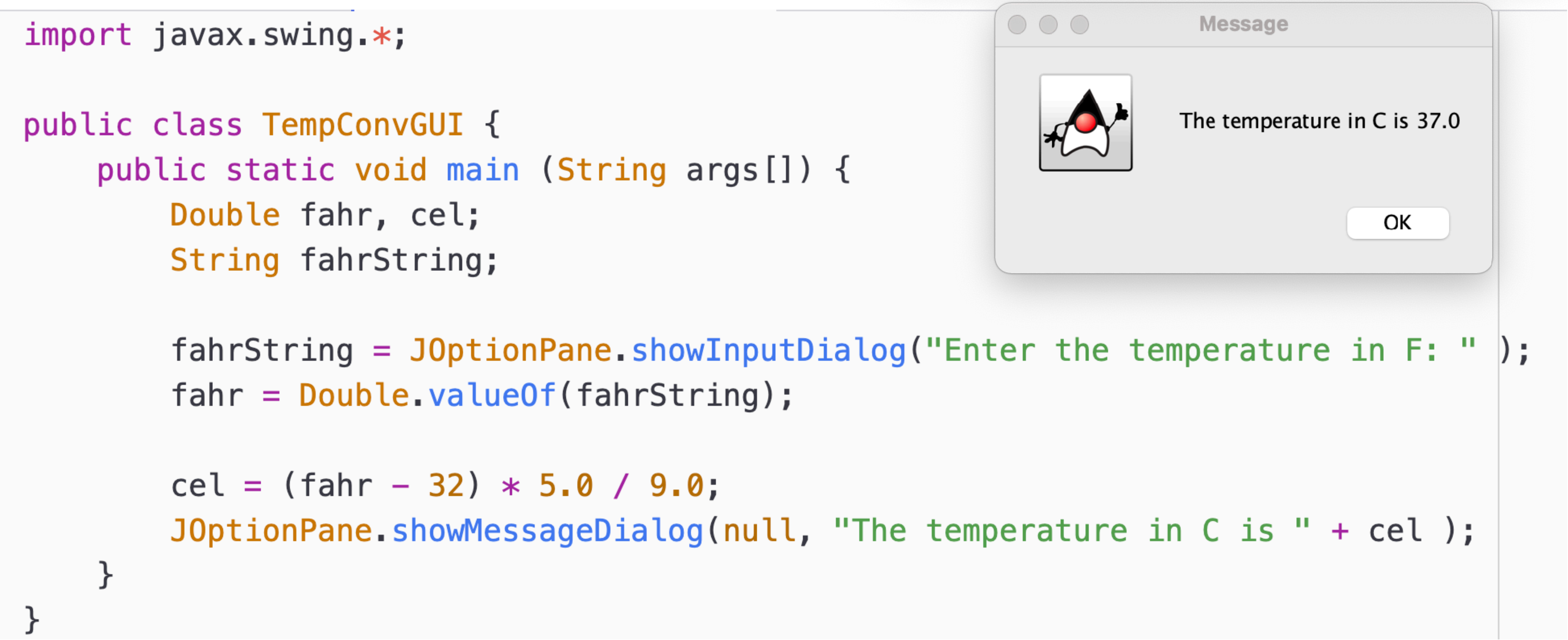
An Aside: Java GUIs

- Java has more built-in support for making GUIs and supporting graphical objects
- Here is a graphical version of our program



Enter the temperature in F:

98.6



```
import javax.swing.*;

public class TempConvGUI {
    public static void main (String args[]) {
        Double fahr, cel;
        String fahrString;

        fahrString = JOptionPane.showInputDialog("Enter the temperature in F: " );
        fahr = Double.valueOf(fahrString);

        cel = (fahr - 32) * 5.0 / 9.0;
        JOptionPane.showMessageDialog(null, "The temperature in C is " + cel );
    }
}
```

The temperature in C is 37.0