COMPSCI 121: INTRODUCTION TO PROGRAMMING

SPRING 2020

WHAT ARE THE GOALS FOR TODAY'S CLASS

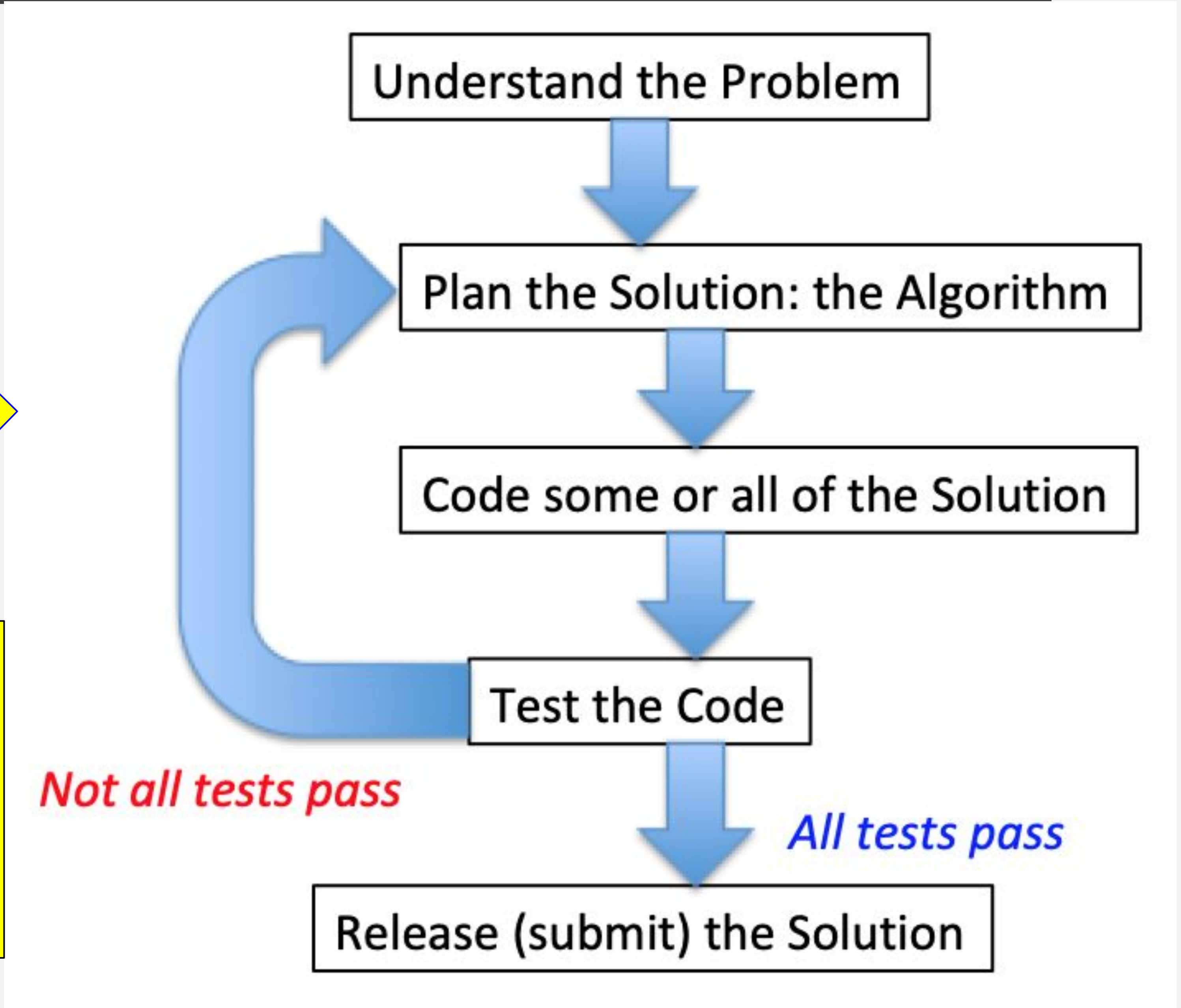
- Introduce you to some java programming constructs.
- Introduce you to problem solving techniques.
- Demo: using jGRASP
- Point out good programming practices.

WHY ARE WE USING JAVA?

- Java is a widely used language in industry.
- Knowledge of Java is in high demand.
- It is a mature language (1995) and is frequently updated.
- Helps you learn other languages.
- Created as a “Software Engineering” language so that large programs can be more easily developed and maintained.
- It is designed to support software best practices.

PROGRAMMING STEPS & PATTERN

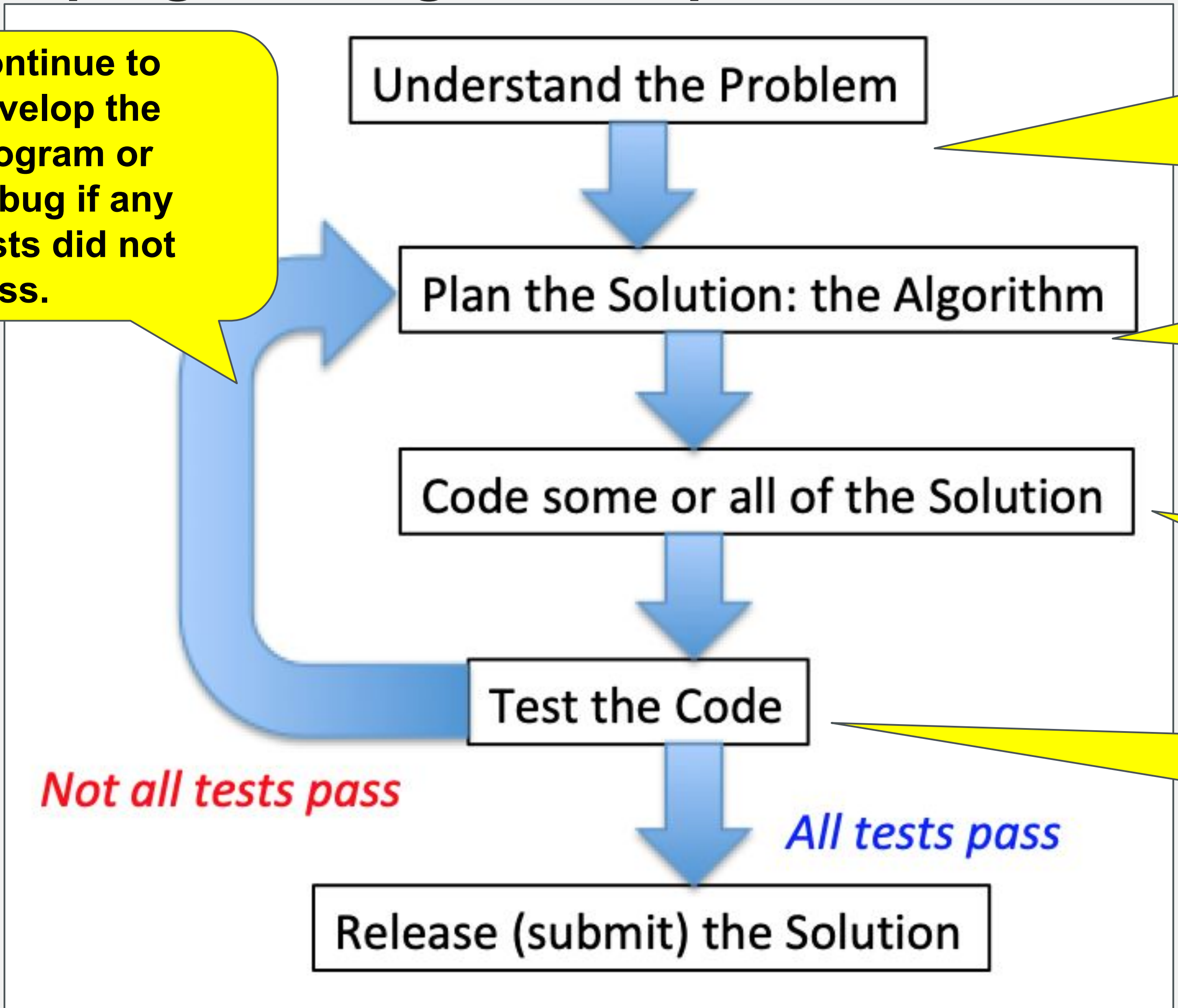
Best to write a little code, test it, debug (find and remove errors) if test(s) failed, or continue to write more code if the test(s) passed.


Understand the Problem
Plan the Solution: the Algorithm
Code some or all of the Solution
Test the Code
Not all tests pass
All tests pass
Release (submit) the Solution

Note:
A computer can only execute one step at a time. Each step must be clearly defined.

PROGRAMMING STEPS

All programming has this pattern:


Understand the Problem
Plan the Solution: the Algorithm
Code some or all of the Solution
Test the Code
Not all tests pass
All tests pass
Release (submit) the Solution

Continue to develop the program or debug if any tests did not pass.

Read and analyze all aspects of the problem the program is to address. Make sure you *understand* what the program should do- its logic. What are the *inputs* and *outputs* to the program?

Write a step-by-step process, an *algorithm*, that will use the input to produce the expected output.

Write code that will *correctly* implement some or all of the algorithm.

***Test* some or all of the program: compare program output to correct output given a set of inputs.**

BAKING ANALOGY

- **Problem:** I want some banana bread.
  - **Input:** The ingredients.
  - **Output:** Warm banana bread to eat!
  - **Test:** Must be edible and must look and taste like banana bread.
- **Algorithm:** A step-by-step plan, like a recipe, for getting from the Input to the Output.
- **Implementation:** in this case, ME!

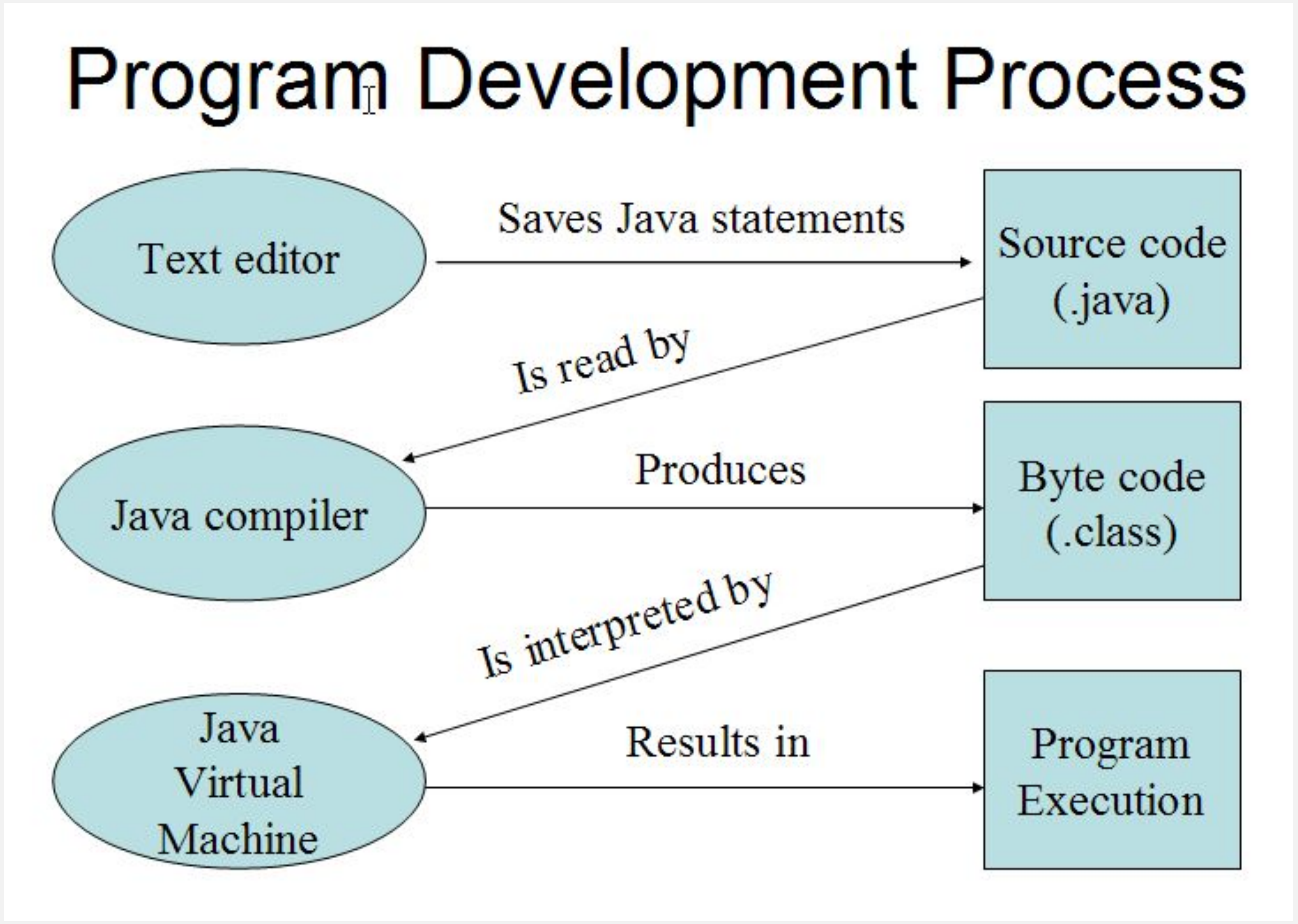
AN ALGORITHM YOU CAN ENJOY!

1. Preheat oven to 350 degrees F (175 degrees C).
2. Lightly grease a 9x5 inch loaf pan.
3. In a large bowl, combine flour, baking soda and salt.
4. In a separate bowl, cream together butter and brown sugar.
5. Stir in eggs and mashed bananas until well blended.
6. Stir banana mixture into flour mixture; stir just to moisten.
7. Pour batter into prepared loaf pan.
8. Bake in preheated oven for 60 to 65 minutes.
9. Bake until a toothpick inserted into center of the loaf comes out clean.
10. Let bread cool in pan for 10 minutes.
11. Turn out onto a wire rack.
12. Test the output!

In this case, there is no way to partially implement the algorithm, so if the test fails, you'll have to start over!

HOW DOES PROGRAMMING IN JAVA WORK?

1. The problem is analyzed and understood.
2. Inputs and outputs defined.
3. An algorithm is designed.
4. Then, Java code is written in one or more source code files.
5. This file is compiled and run.
6. The output is generated.
7. Tests can be run on the output.



PROBLEM STATEMENT & ANALYSIS

Write a program that calculates the annual salary given the hourly wage.

1. What are the inputs?
2. What are the outputs?

THINK - PAIR - SHARE!

DESIGN THE ALGORITHM

Write a program that calculates the annual salary given the hourly wage.

1. What are the inputs? hourly wage
2. What are the outputs? annual salary
3. What is the algorithm (steps) for the program?

THINK - PAIR - SHARE!

DESIGN THE ALGORITHM

3. What is the algorithm (steps) for the program?

1. **Get the hourly rate (for example, from user).**

   hourlyWage = ?

2. **Calculate the annual salary:**

   hoursPerWeek = 40, weeksPerYear = 50

   annSalary = hourlyWage * hoursPerWeek * weeksPerYear

3. **Print the result.**

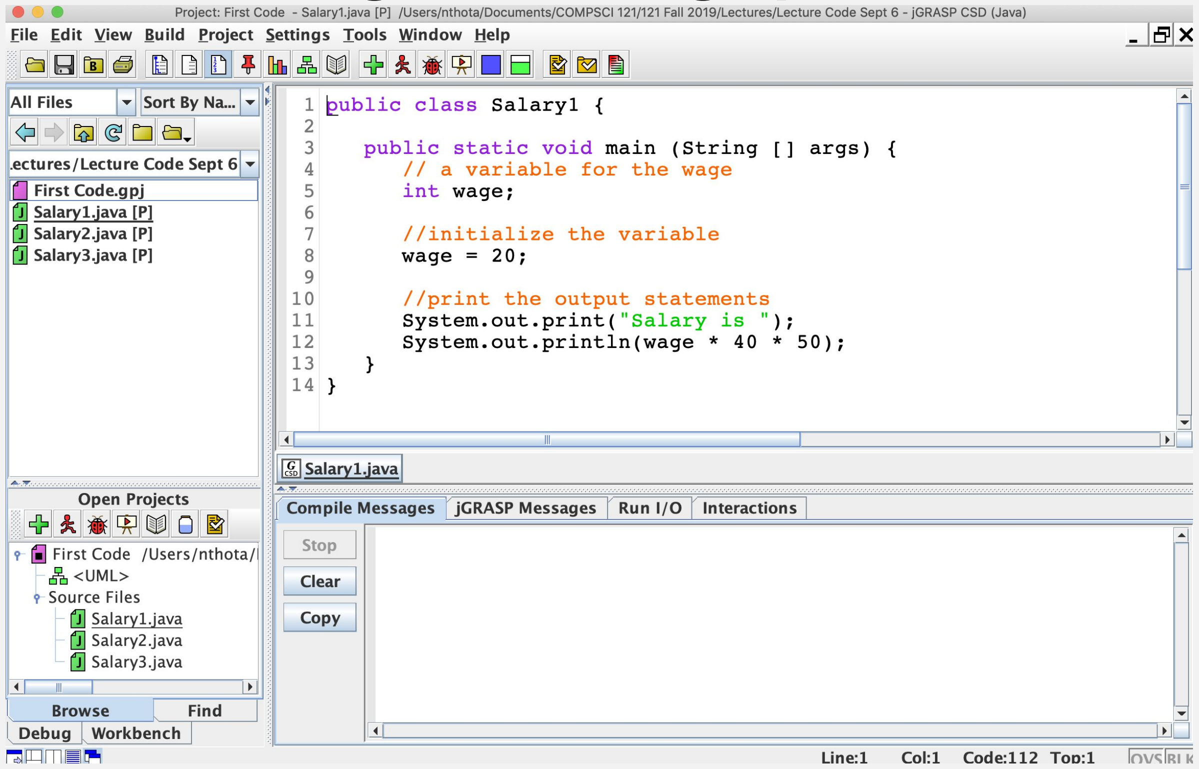
DEMO JGRASP

Saving files in folders

Compiling & running the `Salary` program (from zyBooks).

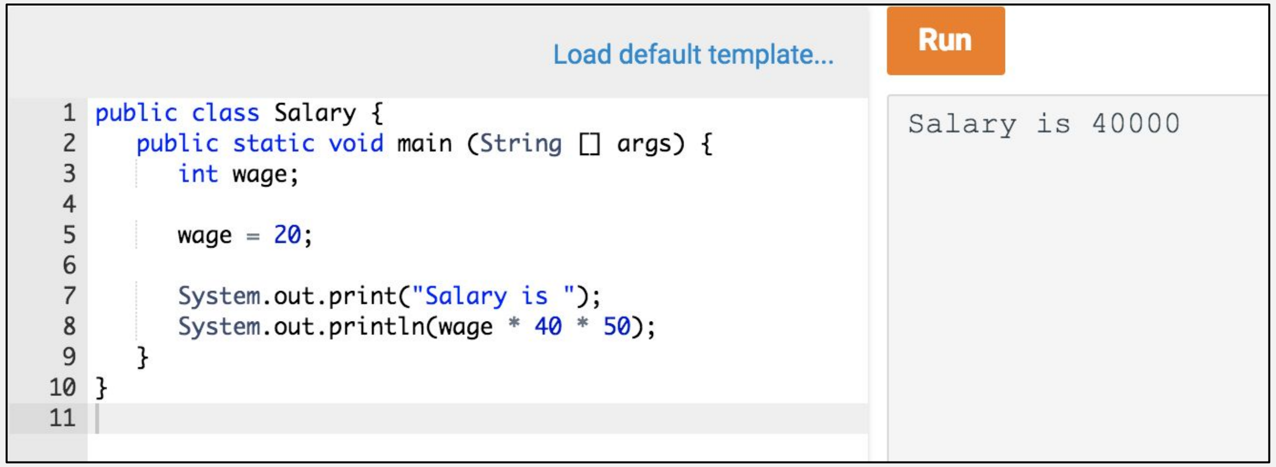
Printing in single/new lines.

Making/removing syntax errors.



Project: First Code - Salary1.java [P] /Users/nthota/Documents/COMPSCI 121/121 Fall 2019/Lectures/Lecture Code Sept 6 - jGRASP CSD (Java)

```java
public class Salary1 {

   public static void main (String [] args) {
      // a variable for the wage
      int wage;

      //initialize the variable
      wage = 20;

      //print the output statements
      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```



```java
public class Salary {
   public static void main (String [] args) {
      int wage;

      wage = 20;

      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```

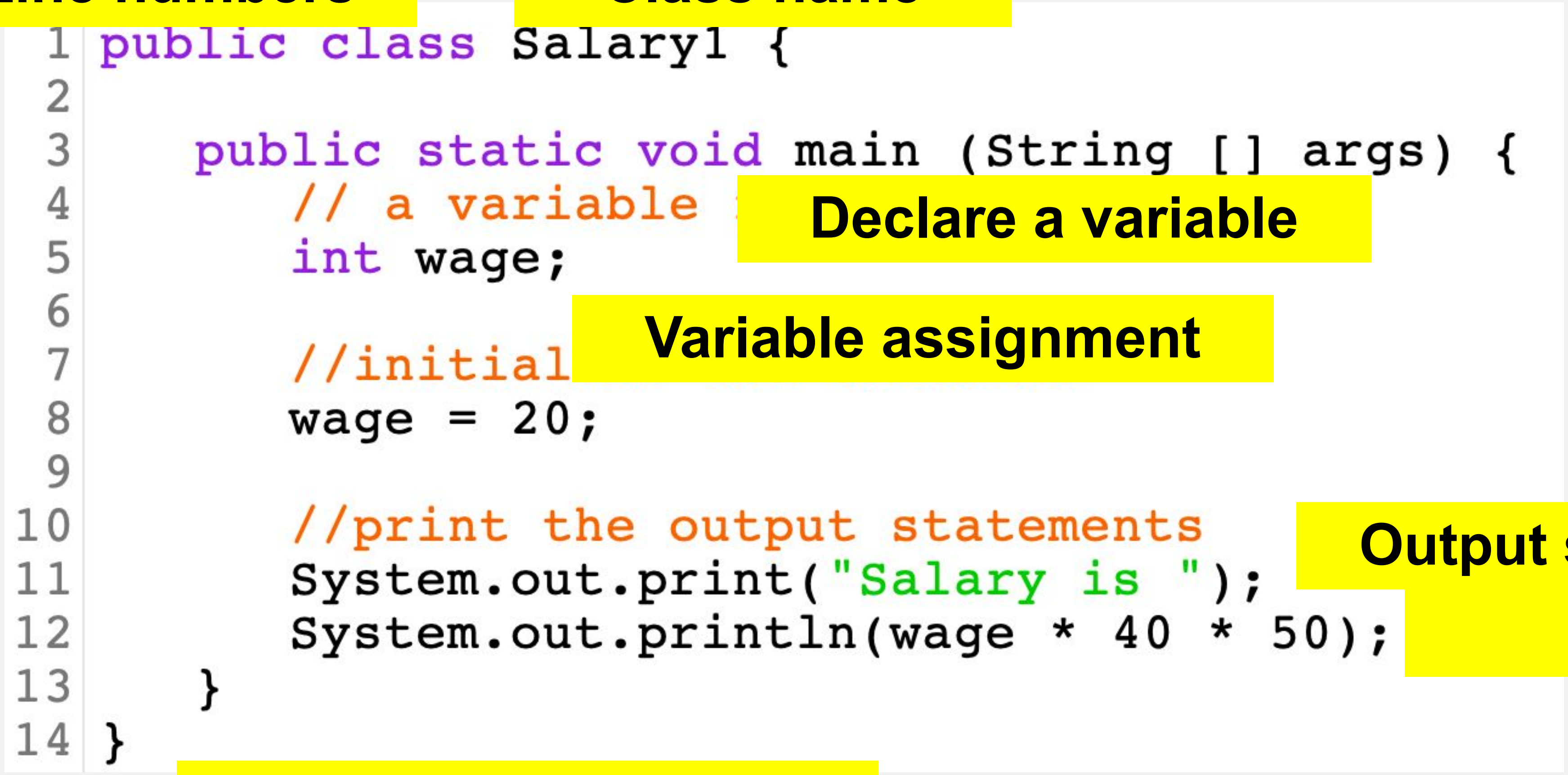
```
Salary is 40000
```

THE JAVA CODE - CHECK FOR ERRORS

DEMO IN JGRASP

Line numbers

Class name



```
 1 public class Salary1 {
 2 
 3    public static void main (String [] args) {
 4       // a variable
 5       int wage;
 6 
 7       //initial
 8       wage = 20;
 9 
10       //print the output statements
11       System.out.print("Salary is ");
12       System.out.println(wage * 40 * 50);
13    }
14 }
```

Declare a variable

Variable assignment

Output statements

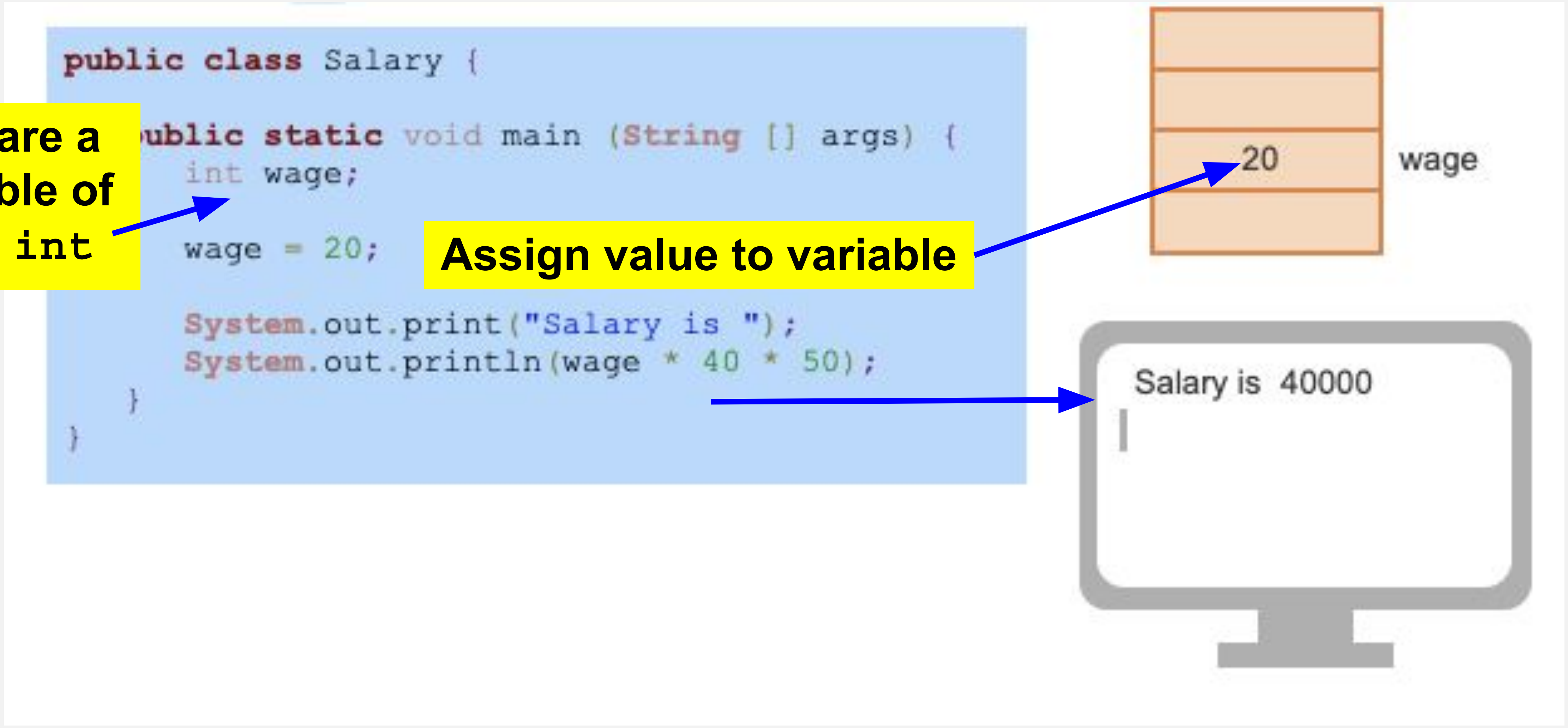
Semi-colons

{ } braces for scope

Note: Code in jGRASP is color-coded

WHAT ARE VARIABLES?

Programs use ***variables*** to refer to data.



```
public class Salary {

   public static void main (String [] args) {
      int wage;

      wage = 20;

      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```

Declare a variable of type `int`

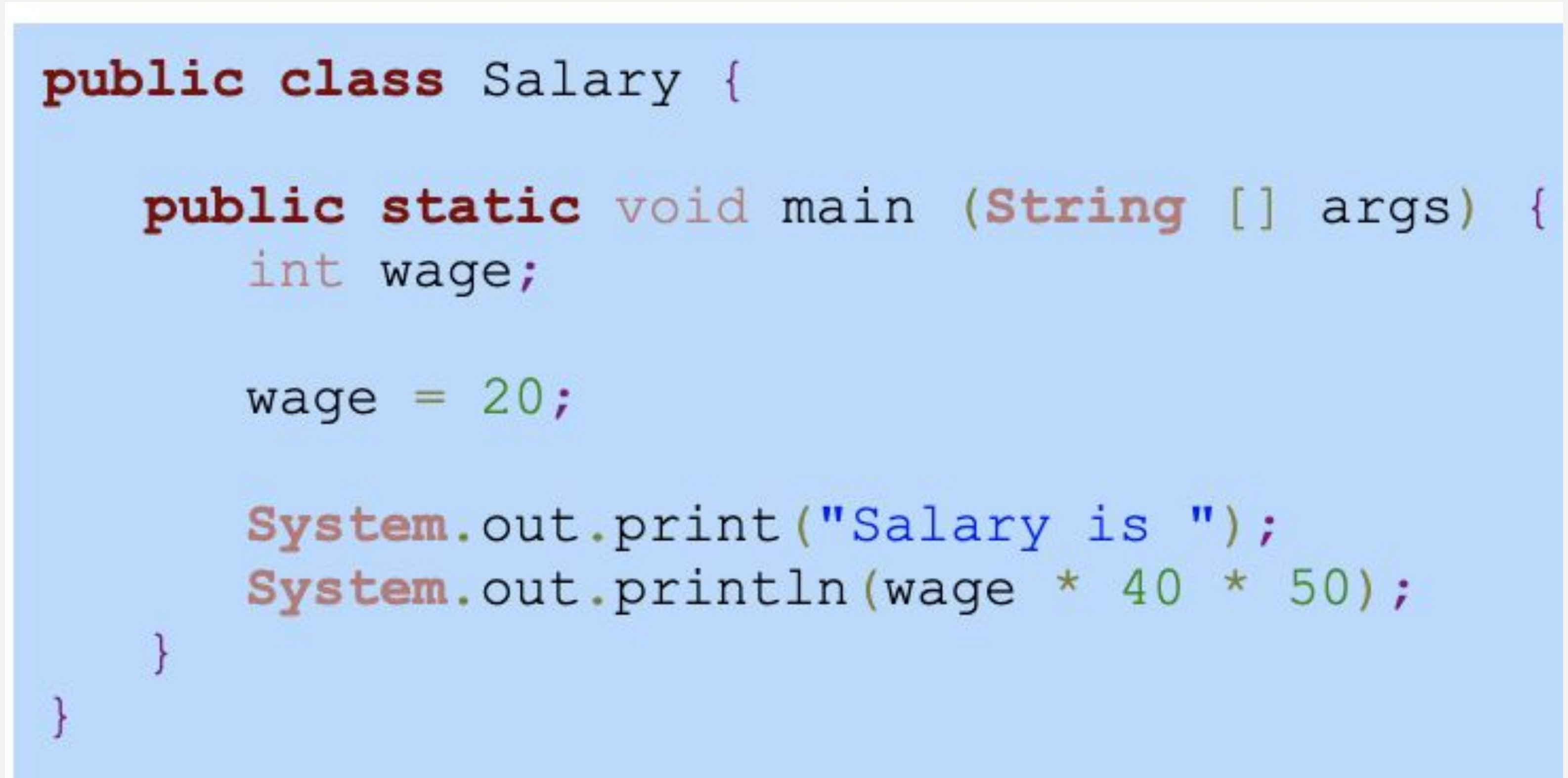
Assign value to variable

```
Salary is  40000
```

CLICKER QUESTION #1

On which line does the program start *when it runs*?

A. On the first line of the program `public class Salary`
B. With the main statement `public static void main`
C. With the print statement `System.out.print`



```
public class Salary {

   public static void main (String [] args) {
      int wage;

      wage = 20;

      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```

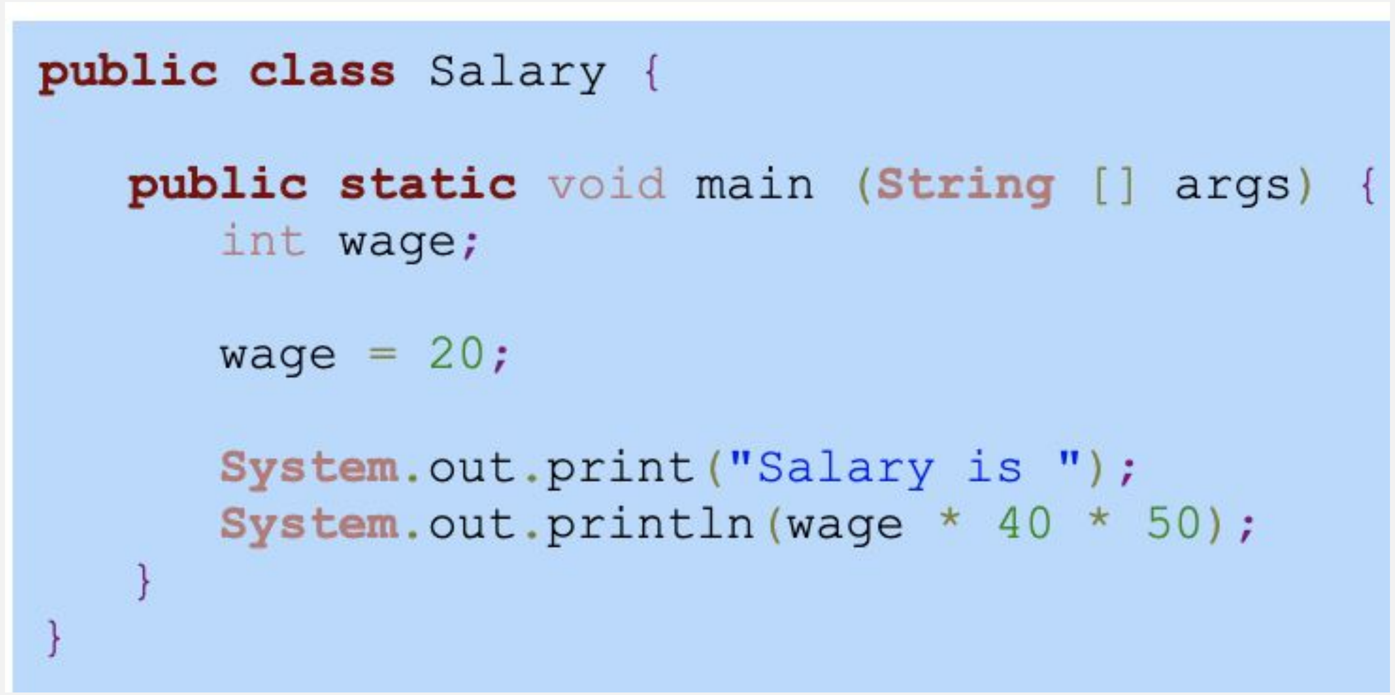
ANSWER CLICKER QUESTION #1

On which line does the program start?

A. On the first line of the program `public class Salary`

B. With the main statement `public static void main`

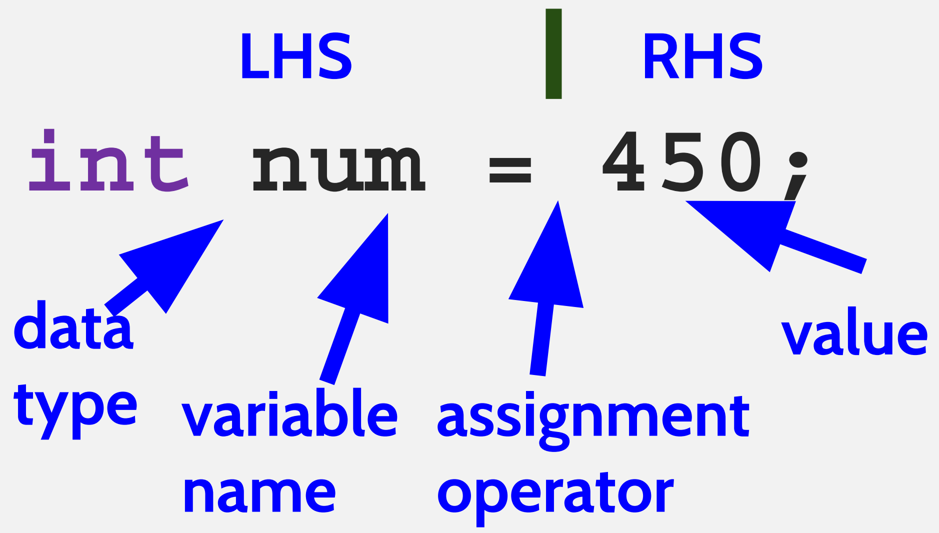
C. With the print statement `System.out.print`



```java
public class Salary {

   public static void main (String [] args) {
      int wage;

      wage = 20;

      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```

HOW DO WE ASSIGN VALUES TO VARIABLES?

Assignment statements: create a variable and assign a value to it.



```
int num = 450;
```

“Declaration” of the variable “`num`” with data type “int”.
It also assigns the value 450 to that variable.

- LHS has to be a **variable**, RHS has to resolve to a **value**.

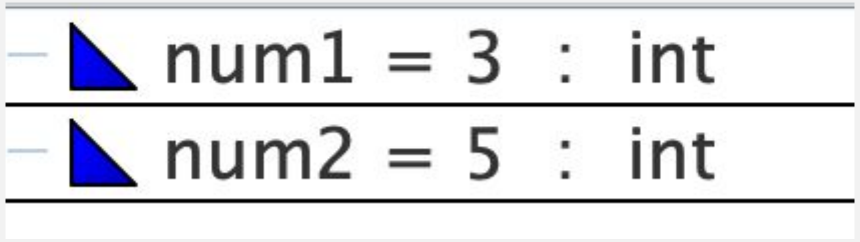
ASSIGNMENT IN JAVA: WHAT HAPPENS IN MEMORY

EXAMPLE:

After the execution of the last statement, the variable "`num1`" now references an integer value of 3.

```
int num1 = 5;
int num2 = num1; // values in num1 and num2 are now equal
num1 = 3; // values in num1 and num2 are no longer equal
```

State diagram



```
num1 = 3 : int
num2 = 5 : int
```

ASSIGNMENT IN JAVA: WHAT HAPPENS IN MEMORY

Notice that you *declare* a variable only once,
but can use it as much as necessary once it's declared.

You cannot re-declare a variable:

```
int num = 450;
```

declaration *OK*

~~int num = 22;~~

re-declaration *Not OK*

You *can* re-assign a variable:

```
int num = 450;
num = 22;
```

re-assignment *OK*

CLICKER QUESTION 2

Which variable holds the highest number?

```
int num = 14;
int num2 = (num + 1);
int num3 = num2;
num2 = num3 + 3;
num = 17;
```

A. num
B. num2
C. num3

ANSWER CLICKER QUESTION 2

Which variable holds the highest number?

```
1. int num = 14;
2. int num2 = (num + 1);
3. int num3 = num2;
4. num2 = num3 + 3;
5. num = 17;
```

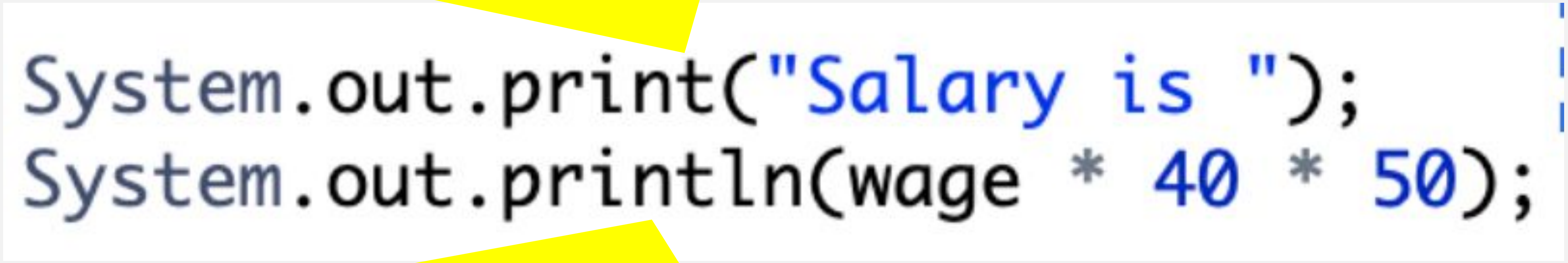
A. `num = 17`

B. `num2 = 18`

C. `num3 = 15`

HOW DID WE SHOW THE OUTPUT TO THE USER?

Uses "" to output a string literal.
Multiple output statements continue printing on the same output line.



```
System.out.print("Salary is ");
System.out.println(wage * 40 * 50);
```

Uses "" to output a string literal.
Starts a new output line after the outputted values, called a newline.

CLICKER QUESTION 3

Given this code (assume variables have been declared):

```
hourlyWage = 20, hoursPerWeek = 40,  weeksPerYear = 50;
annSalary = hourlyWage * hoursPerWeek *  weeksPerYear;
```

Which one does *not* print 40000?

```
A.  System.out.print(annSalary);
B.  System.out.print(hourlyWage * hoursPerWeek *
    weeksPerYear);
C.  System.out.print(“annSalary”);
D.  System.out.print(20 * 40 *  50);
```

CLICKER QUESTION 3

Given this code (assume variables have been declared):

```
hourlyWage = 20, hoursPerWeek = 40,  weeksPerYear = 50;
annSalary = hourlyWage * hoursPerWeek *  weeksPerYear;
```

Which one does *not* print 40000?

```
A.  System.out.print(annSalary);
B.  System.out.print(hourlyWage * hoursPerWeek *
    weeksPerYear);
C.  System.out.print(“annSalary”); prints literal
D.  System.out.print(20 * 40 *  50);
```

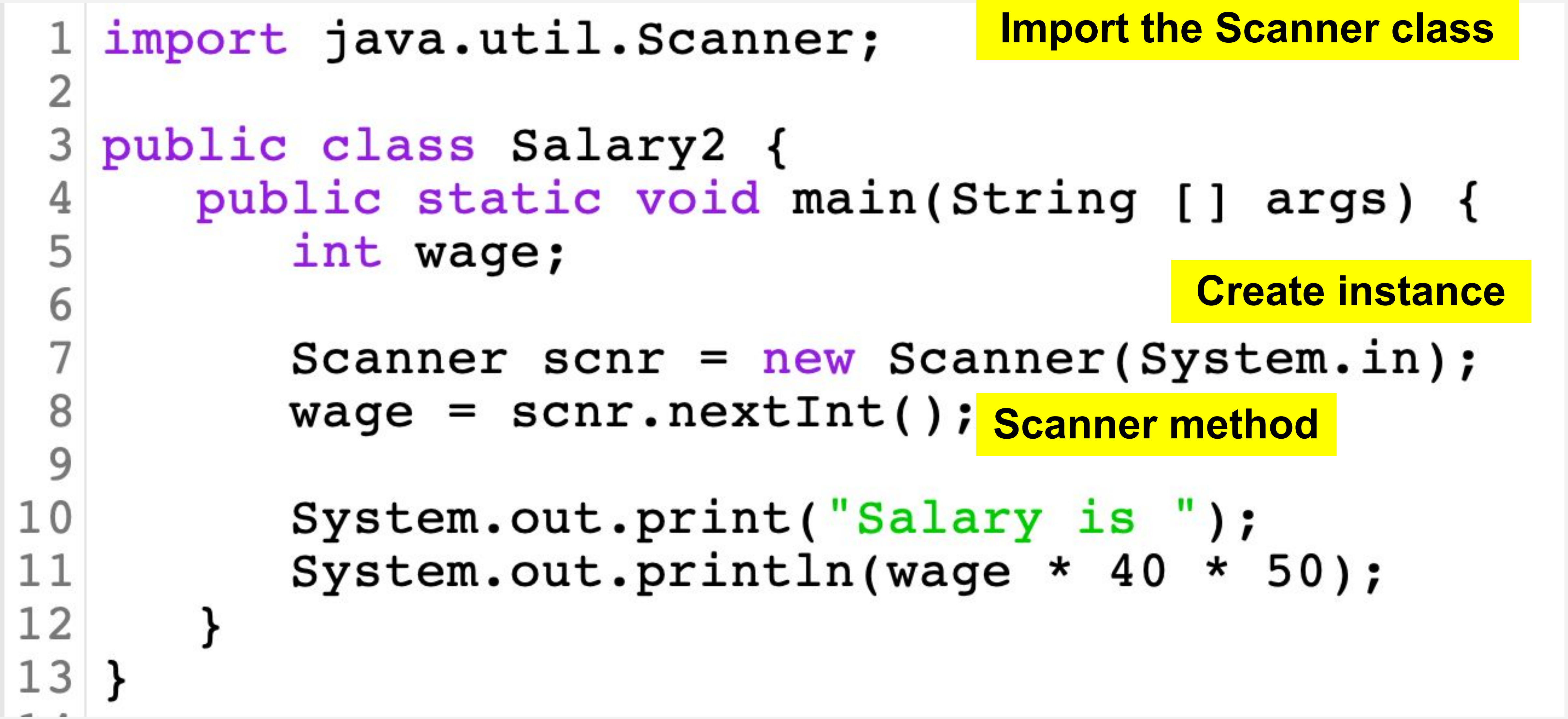
PROBLEM: HOW DO WE GET USER INPUT FOR OUR PROGRAM?

For our `Salary` program, we want the user to input the wage from the keyboard.

Solution: we use the special Java text parser called `Scanner`.
Steps:

1. Create a Scanner object: Scanner scnr = new Scanner(System.in);
   `System.in` corresponds to keyboard input.
2. Given Scanner object `scnr`, get an input value and assign to a variable with `scnr.nextInt()`
   `scnr.nextInt()` is a function (method) that gets the input value of type integer (int)

USING SCANNER TO GET USER INPUT FROM KEYBOARD: Demo jGRASP



```java
import java.util.Scanner;

public class Salary2 {
   public static void main(String [] args) {
      int wage;

      Scanner scnr = new Scanner(System.in);
      wage = scnr.nextInt();

      System.out.print("Salary is ");
      System.out.println(wage * 40 * 50);
   }
}
```

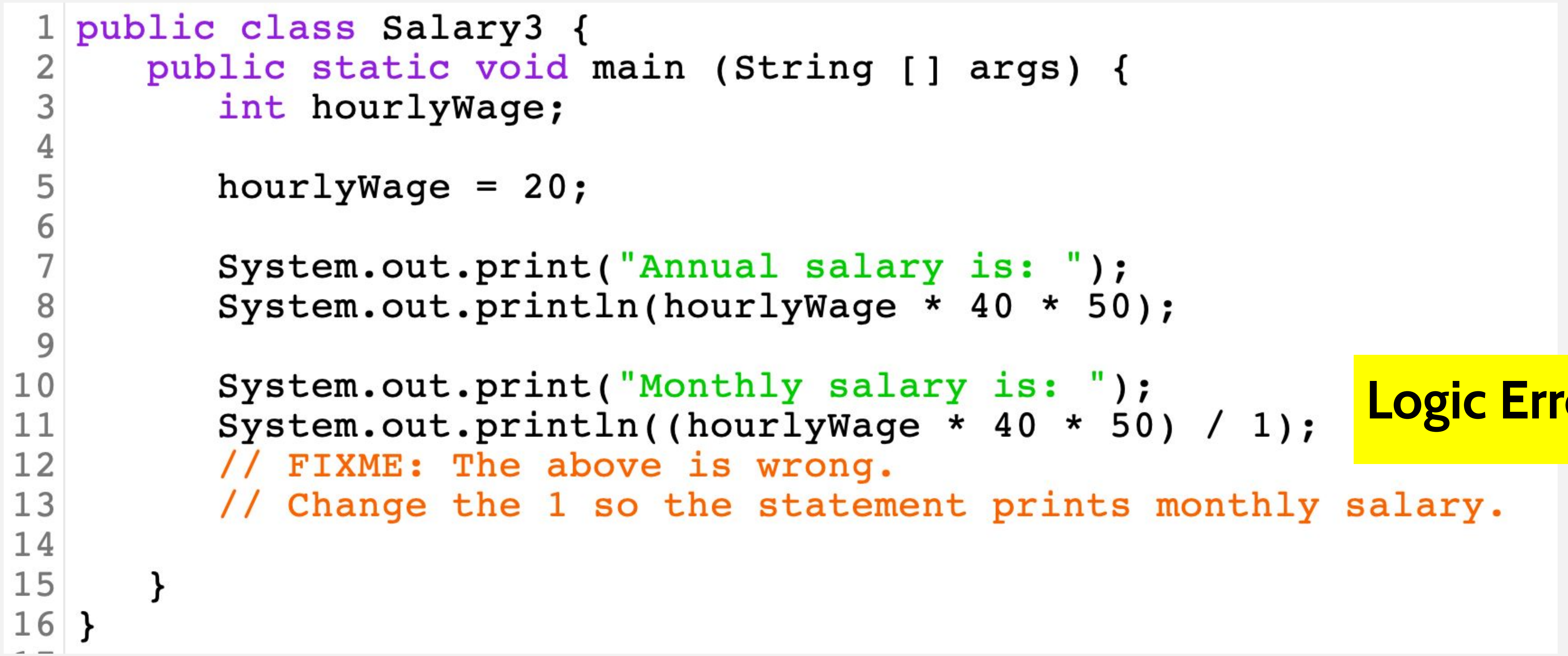
Import the Scanner class

Create instance

Scanner method

TESTING FOR LOGIC ERRORS

Now let's calculate the monthly salary.
What if our program works but the output is wrong?



```java
public class Salary3 {
   public static void main (String [] args) {
      int hourlyWage;

      hourlyWage = 20;

      System.out.print("Annual salary is: ");
      System.out.println(hourlyWage * 40 * 50);

      System.out.print("Monthly salary is: ");
      System.out.println((hourlyWage * 40 * 50) / 1);
      // FIXME: The above is wrong.
      // Change the 1 so the statement prints monthly salary.

   }
}
```

Logic Error

GRADING CRITERIA OF PROGRAMMING PROJECTS

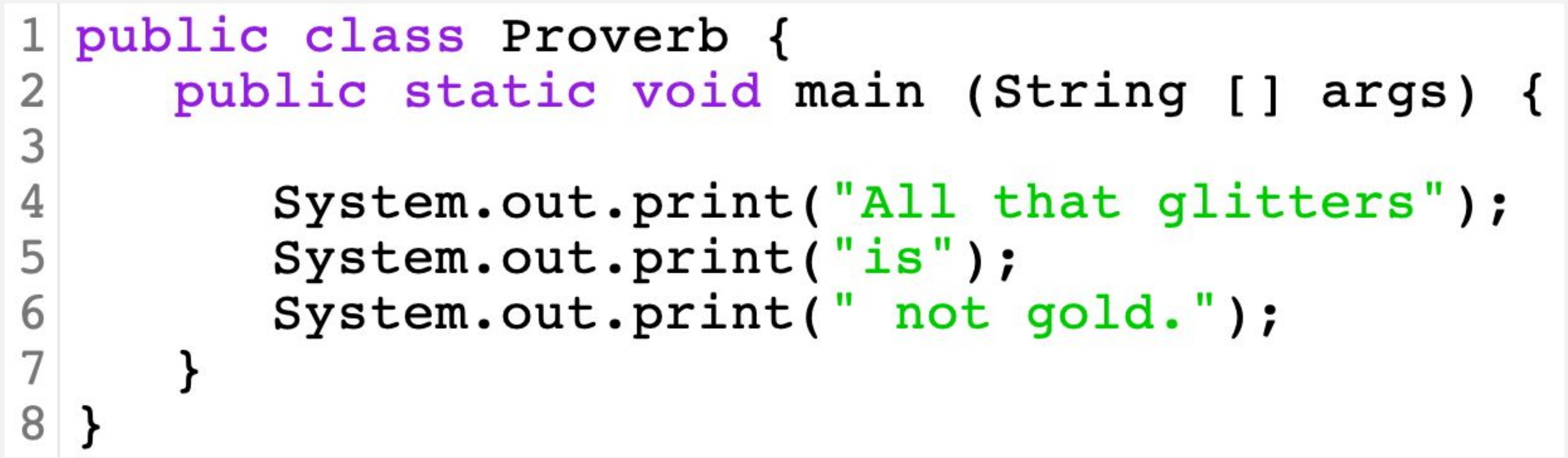
Code Style: follows good style (as defined in the course material). This means your code is readable to you and to others. This is professional "*best practice*".

Code Compiles: A program that doesn't compile cannot run and is not a successful program. This program does NOT get credit.

Logical Correctness: The code is logically correct if it runs and produces the correct output.

CLICKER QUESTION 4

What is the output of running this file?



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

        System.out.print("All that glitters");
        System.out.print("is");
        System.out.print(" not gold.");
    }
}
```

```
A. All that glittersis not gold.
B. All that glitters is not gold.
C. All that glitters isnot gold.
D. All that glitters is  not gold.
```

CLICKER QUESTION 4 ANSWER

What is the output of running this file?

```java
public class Proverb {
    public static void main (String [] args) {

        System.out.print("All that glitters");
        System.out.print("is");
        System.out.print(" not gold.");
    }
}
```

```
A. All that glittersis not gold.
B. All that glitters is not gold.
C. All that glitters isnot gold.
D. All that glitters is  not gold.
```

Watch white spaces!

SOME GOOD PROGRAMMING PRACTICES

1. **Create a good solution (algorithm) first, before you start coding.**
2. **Compile after writing only a few lines of code, rather than writing tens of lines and then compiling.**
3. **Make incremental changes— Don't try to change everything at once.**
4. **Focus on fixing just the first error reported by the compiler, and then re-compiling.**
5. **When you don't find an error at a specified line, look to previous lines.**
6. **Be careful not type the number "1" or a capital I in `System.out.println`.**
7. **Do not type numbers directly in the output statements; use the variables.**

Week 1 TODO List:

1. Register your iClicker in Moodle.
2. Install JAVA and jGRASP.
3. Complete zyBook chapter 1 exercises.
4. Attend lab on Friday with your laptop.
5. Read the course documents (Moodle).
6. Give your consent to use codePost.
7. Complete the Orientation Quiz in Moodle.
8. Ask questions in Piazza.