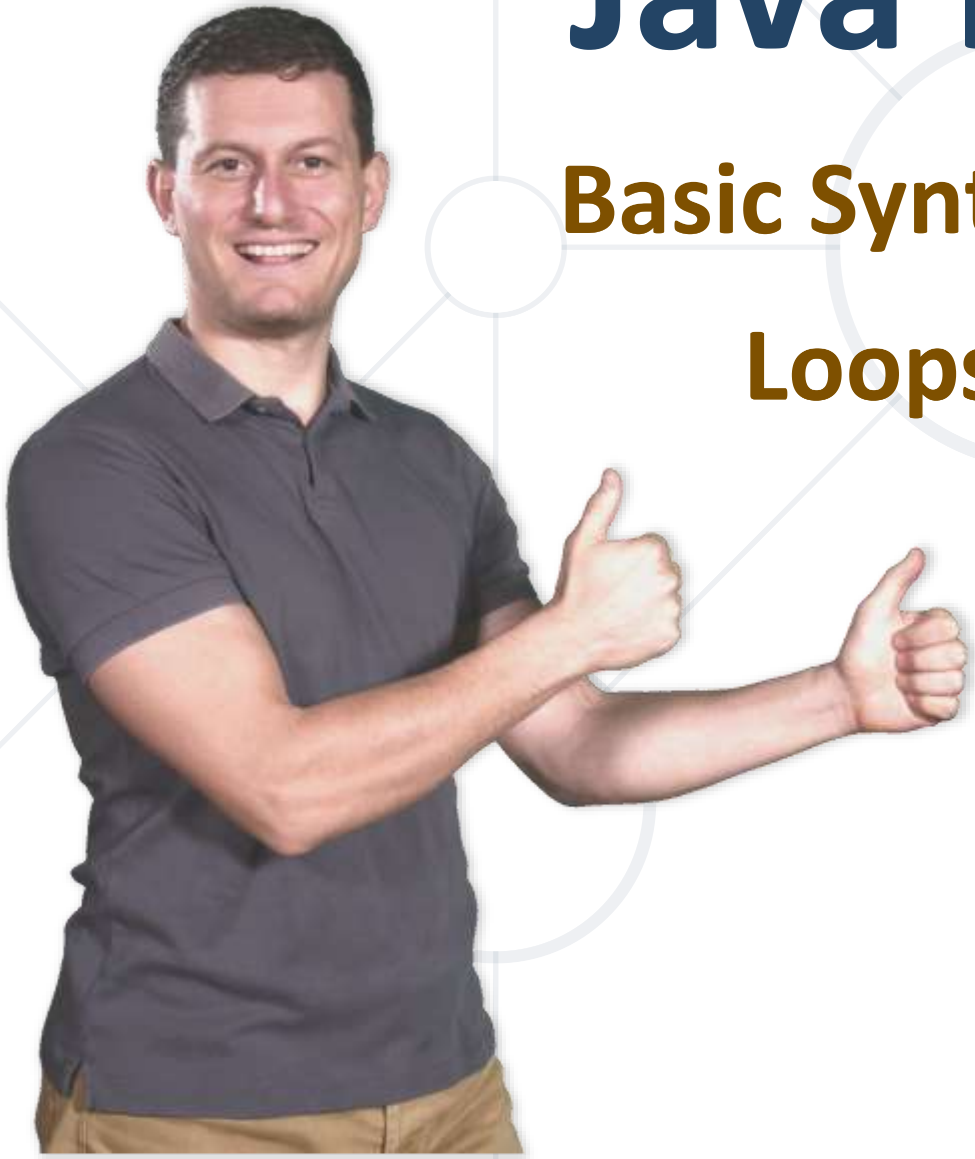
Java Foundation

Basic Syntax , I/O, Conditions, Loops and Debugging

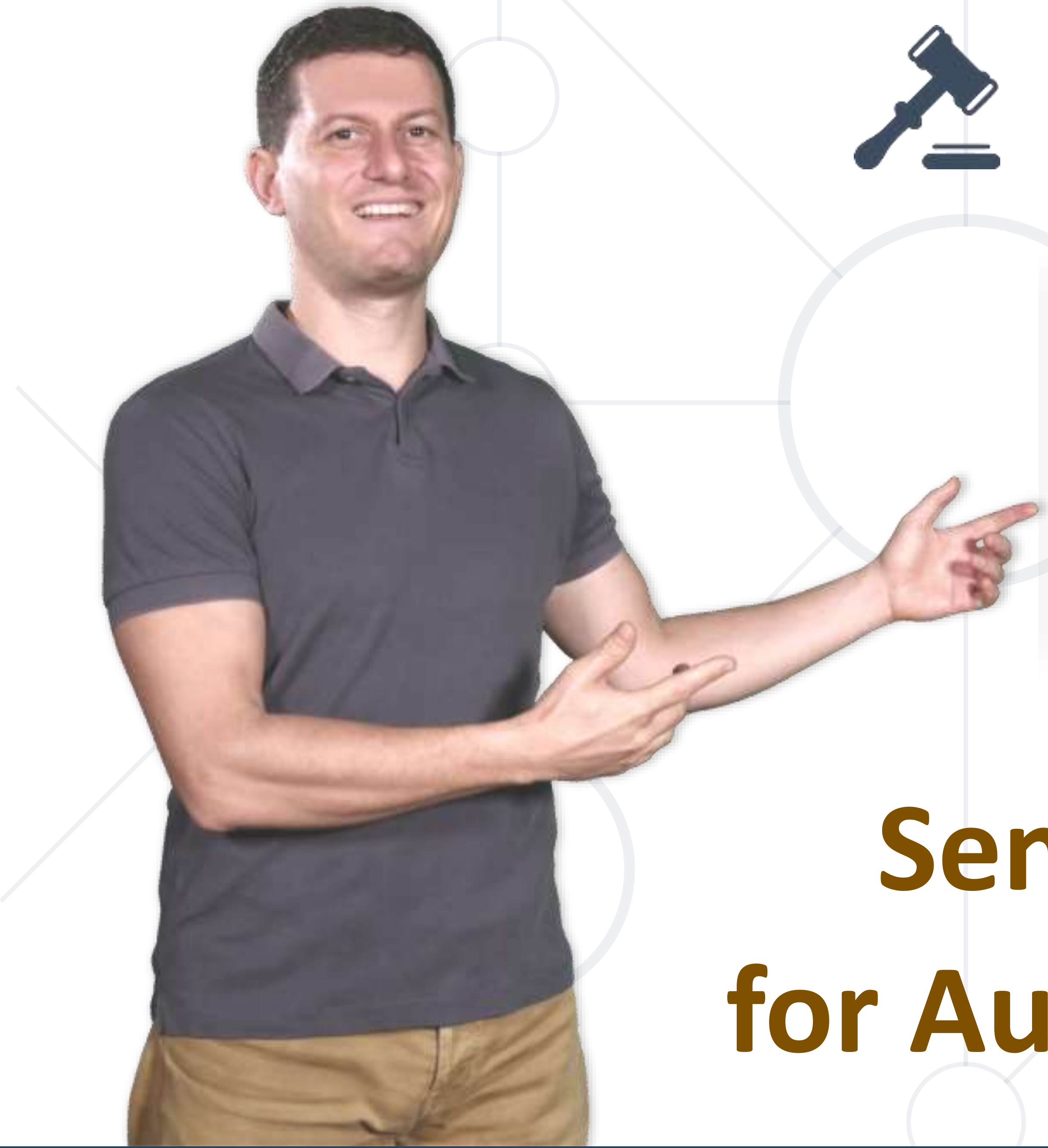


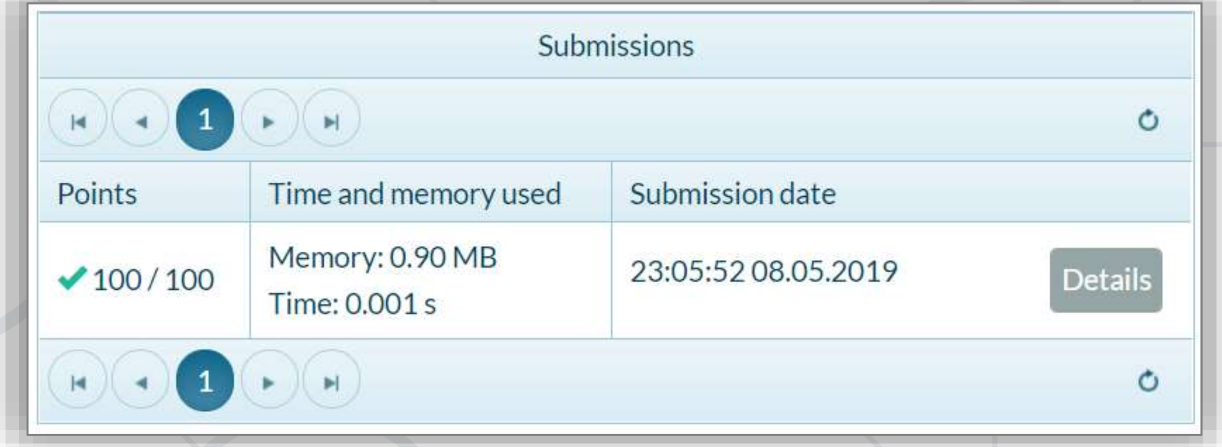
SoftUni

George Georgiev

- Technical trainer @ SoftUni
  - C++, C#, Java, Data Structures and others
- Developer @ VirtualRacingSchool.com
  - Java, JavaScript, C++
- Experience
  - 6+ years training, 12+ years coding
- Wrote a driving simulator in high school (DriveFreeZ award)
- Played around with OpenGL, Bullet Physics SDK, WinRT

The Judge System



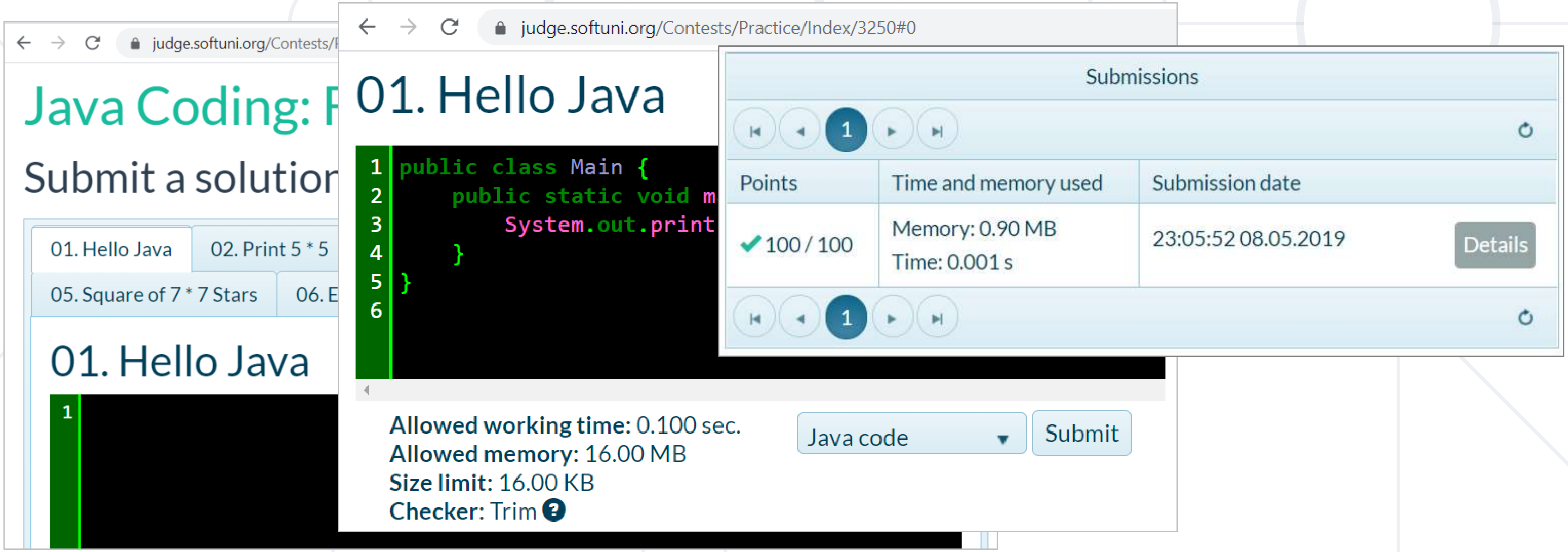

Submissions

Points	Time and memory used	Submission date	
✓ 100 / 100	Memory: 0.90 MB Time: 0.001 s	23:05:52 08.05.2019	Details

Sending your Solutions for Automated Evaluation

Testing the Program in the Judge System

- Test your code online in the SoftUni **Judge system**: https://judge.softuni.org/Contests/3294



judge.softuni.org/Contests/Practice/Index/3250#0

01. Hello Java

```
public class Main {
    public static void m
        System.out.print
    }
}
```

Allowed working time: 0.100 sec.
Allowed memory: 16.00 MB
Size limit: 16.00 KB
Checker: Trim

Submissions

Points	Time and memory used	Submission date	
✓ 100 / 100	Memory: 0.90 MB Time: 0.001 s	23:05:52 08.05.2019	Details

Table of Contents

Introduction and Basic Syntax

Java – Introduction

SoftUni

- Java is modern, flexible, general-purpose programming language
- Object-oriented by nature, statically-typed, compiled

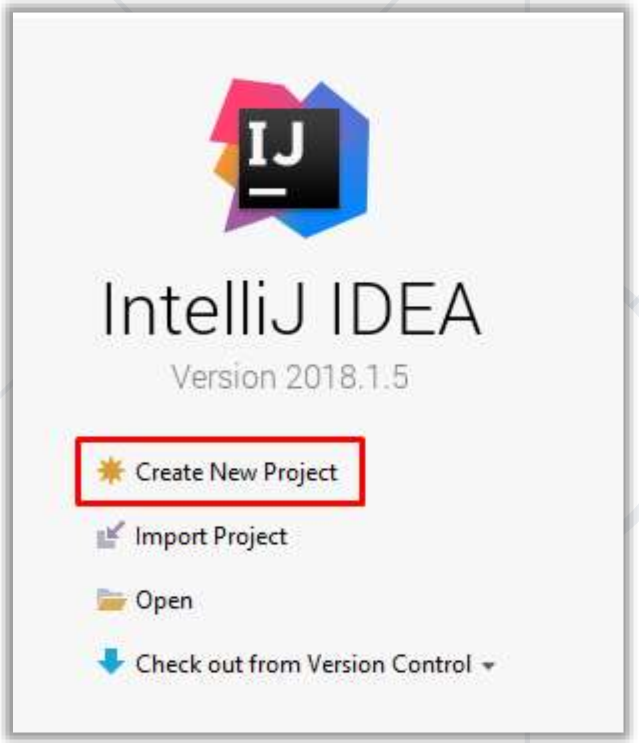
```
static void main(String[] args) {
    //Source Code
}
```

Program starting point

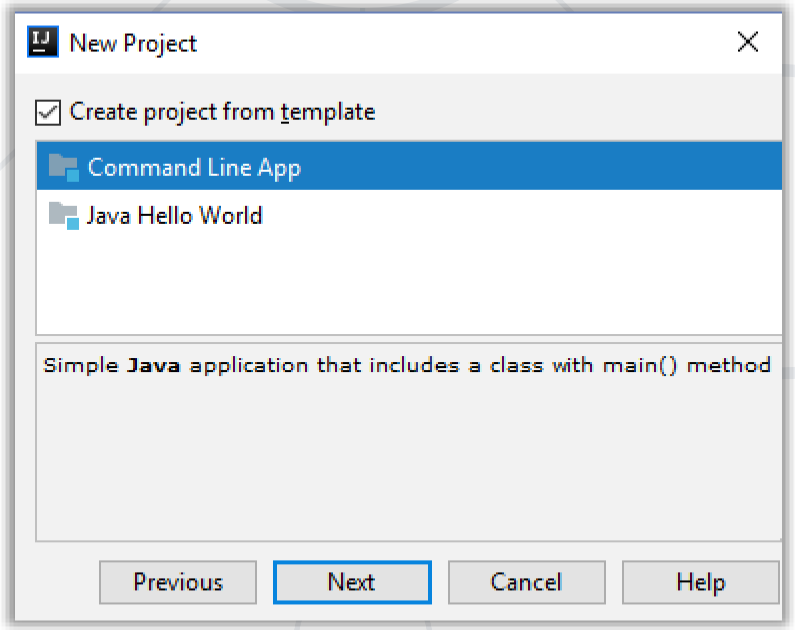
- In this course will use [Java Development Kit (JDK) 12]

Using Intellij Idea

- **Intellij Idea** is powerful IDE for Java and other languages
- Create a project



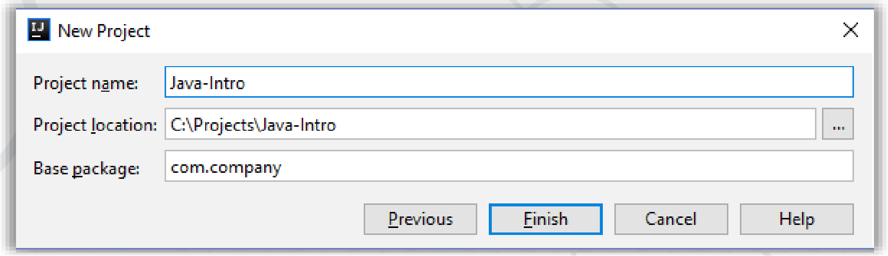
IntelliJ IDEA
Version 2018.1.5



New Project

☑ Create project from template

Simple Java application that includes a class with main() method



New Project

Project name:	Java-Intro
Project location:	C:\Projects\Java-Intro
Base package:	com.company

Declaring Variables

- Defining and Initializing variables

```
{data type / var} {variable name} = {value};
```

- Example:

```
int number = 5;
```

Variable name

Data type

Variable value

Console I/O

Reading from and Writing to the Console

Reading from the Console

- We can **read/write** to the console, using the **Scanner** class
- Import the **java.util.Scanner** class

```
import java.util.Scanner;
…
Scanner sc = new Scanner(System.in);
```

- Reading input from the console using

```
String name = sc.nextLine();
```

Returns **string**

Converting Input from the Console

- scanner.nextLine() returns a String
- Convert the string to number by **parsing**:

```
import java.util.Scanner;

…

Scanner sc = new Scanner(System.in);

String name = sc.nextLine();

int age = Integer.parseInt(sc.nextLine());

double salary = Double.parseDouble(sc.nextLine());
```

Printing to the Console

- We can **print** to the console, using the **`System`** class
- Writing output to the console:
  - **`System.out.print()`**
  - **`System.out.println()`**

```java
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.println("Hi, " + name);
// Name: George
// Hi, George
```

Using Print Format

- Using **format** to print at the console
- Examples:

```
String name = "George";
int age = 5;
System.out.printf("Name: %s, Age: %d", name, age);
// Name: George, Age: 5
```

Placeholder **%s** stands for string and corresponds to **name**

Placeholder **%d** stands for integer number and corresponds to **age**

Formatting Numbers in Placeholders

- D – format number to certain digits with leading zeros
- F – format floating point number with certain digits after the decimal point
- Examples:

```
int percentage = 55;
double grade = 5.5334;
System.out.printf("%03d", percentage);   // 055
System.out.printf("%.2f", grade);        // 5.53
```

Using String.format

- Using **String.format** to create a string by pattern
- Examples:

```
String name = "George";
int age = 5;
String result = String.format("Name: %s,
                              Age: %d", name, age);
System.out.println(result);
//Name: George, Age 5
```

Problem: Student Information

- You will be given 3 input lines:
  - Student Name, Age and Average Grade
- Print the input in the following format:
  - "Name: {name}, Age: {age}, Grade {grade}"
  - Format the grade to 2 decimal places

```
John
15
5.40
```

```
Name: John, Age: 15, Grade: 5.40
```

Solution: Student Information

```
import java.util.Scanner;

…

Scanner sc = new Scanner(System.in);
String name = sc.nextLine();
int age = Integer.parseInt(sc.nextLine());
double grade = Double.parseDouble(sc.nextLine());


System.out.printf("Name: %s, Age: %d, Grade: %.2f",
                                                    name, age, grade);
```

Comparison Operators

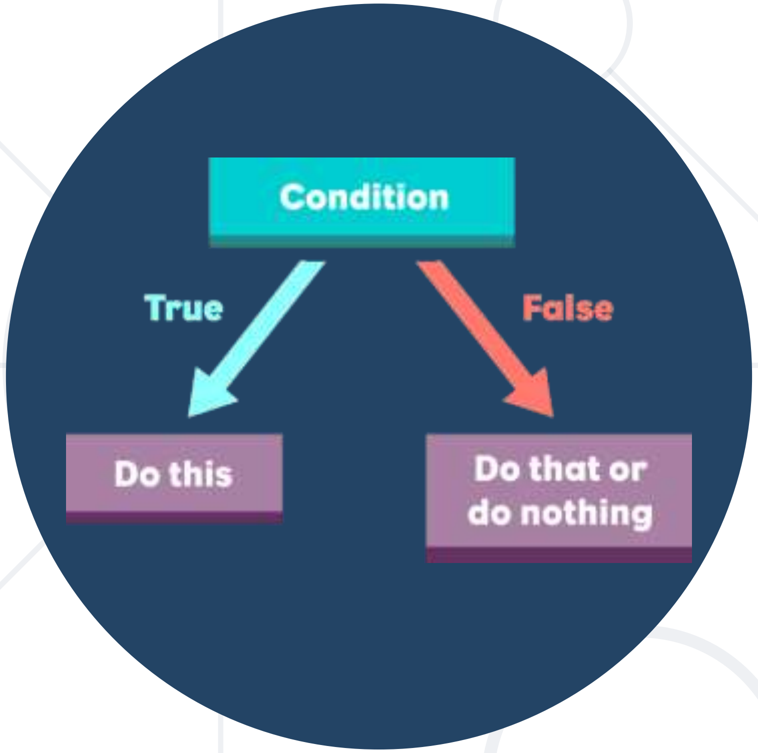
Comparison Operators

Operator	Notation in Java
Equals	==
Not Equals	!=
Greater Than	>
Greater Than or Equals	>=
Less Than	<
Less Than or Equals	<=

Comparing Numbers

- Values can be compared:

```
int a = 5;
int b = 10;
System.out.println(a < b);        // true
System.out.println(a > 0);        // true
System.out.println(a > 100);      // false
System.out.println(a < a);        // false
System.out.println(a <= 5);       // true
System.out.println(b == 2 * a);   // true
```



The if-else Statement

Implementing Control-Flow Logic

The `if` Statement

- The simplest conditional statement
  - Test for a condition
- Example: Take as an input a grade and check if the student has passed the exam (grade >= 3.00)

```
double grade = Double.parseDouble(sc.nextLine());
if (grade >= 3.00) {
   System.out.println("Passed!");
}
```

In Java the opening bracket stays on the same line

The `if-else` Statement

- Executes **one branch** if the condition is **true** and **another**, if it is **false**
- Example: **Upgrade** the last example, so it prints "**Failed**!", if the mark is lower than 3.00:

The **else** keyword stays on a new line

```
if (grade >= 3.00) {
   System.out.println("Passed!");
} else {
  // TODO: Print the message
}
```

Problem: I Will be Back in 30 Minutes

- Write a program that reads hours and minutes from the console and calculates the time after 30 minutes
  - The hours and the minutes come on separate lines
- Example:

```
1            0            23
46  ➡ 2:16   01  ➡ 0:31   59  ➡ 0:29

11           12           11
08  ➡ 11:38  49  ➡ 13:19  32  ➡ 12:02
```

Solution: I Will be Back in 30 Minutes (1)

```java
int hours = Integer.parseInt(sc.nextLine());
int minutes = Integer.parseInt(sc.nextLine()) + 30;

if (minutes > 59) {
  hours += 1;
  minutes -= 60;
}
// Continue on the next slide
```

Solution: I Will be Back in 30 Minutes (2)

SoftUni Foundation

```
if (hours > 23) {
  hours = 0;
}
if (minutes < 10) {
  System.out.printf("%d:%02d%n", hours, minutes);
} else {
  System.out.printf("%d:%d", hours, minutes);
}
```

%n goes on the next line

The Switch-Case Statement

Simplified if-else-if-else

The switch-case Statement

- Works as sequence of **`if-else`** statements
- Example: read input a number and print its corresponding month:

```
int month = Integer.parseInt(sc.nextLine());
switch (month) {
  case 1: System.out.println("January"); break;
  case 2: System.out.println("February"); break;
  // TODO: Add the other cases
  default: System.out.println("Error!"); break;
}
```

Problem: Foreign Languages

- By given country print its typical language:
  - English -> England, USA
  - Spanish -> Spain, Argentina, Mexico
  - other -> unknown

```
England  ➡  English
Spain    ➡  Spanish
```

Solution: Foreign Languages

```java
//TODO: Read the input
switch (country) {
  case "USA":
  case "England": System.out.println("English"); break;
  case "Spain":
  case "Argentina":
  case "Mexico": System.out.println("Spanish"); break;
  default: System.out.println("unknown"); break;
}
```

&&

Logical Operators

Writing More Complex Conditions

Logical Operators

- Logical operators give us the ability to write multiple conditions in one `if` statement
- They return a boolean value and compare boolean values

Operator	Notation in Java	Example
Logical NOT	!	!false -> true
Logical AND	&&	true && false -> false
Logical OR	||	true || false -> true

Problem: Theatre Promotions

SoftUni Foundation

- A theatre has the following ticket prices according to the age of the visitor and the type of day. If the age is < 0 or > 122, print "Error!":

Day / Age	0 <= age <= 18	18 < age <= 64	64 < age <= 122
Weekday	12$	18$	12$
Weekend	15$	20$	15$
Holiday	5$	12$	10$

```
Weekday
42
```

```
18$
```

```
Holiday
-12
```

```
Error!
```

Solution: Theatre Promotions (1)

```java
String day = sc.nextLine().toLowerCase();
int age = Integer.parseInt(sc.nextLine());
int price = 0;
if (day.equals("weekday")) {
  if ((age >= 0 && age <= 18) || (age > 64 && age <= 122)) {
    price = 12;
  }
  // TODO: Add else statement for the other group
}
// Continue…
```

Solution: Theatre Promotions (2)

```java
else if (day.equals("weekend")) {
  if ((age >= 0 && age <= 18) || (age > 64 && age <= 122)) {
    price = 15;
  } else if (age > 18 && age <= 64) {
    price = 20;
  }
} // Continue…
```

Solution: Theatre Promotions (3)

```java
else if (day.equals("holiday")){
  if (age >= 0 && age <= 18)
    price = 5;
  // TODO: Add the statements for the other cases
}
if (age < 0 || age > 122)
  System.out.println("Error!");
else
  System.out.println(price + "$");
```

Loops

Code Block Repetition

Loop: Definition

- A **loop** is a control statement that repeats the execution of a block of statements. The loop can:
  - `for` loop
    - Execute a code block a fixed number of times
  - `while` and `do…while`
    - Execute a code block while a given condition returns true

For-Loops

Managing the Count of the Iteration

For-Loops

SoftUni Foundation

- The for loop executes statements a fixed number of times:

Initial value

End value

Increment

Loop body

```
for (int i = 1; i <= 10; i++) {
    System.out.println("i = " + i);
}
```

Executed at each iteration

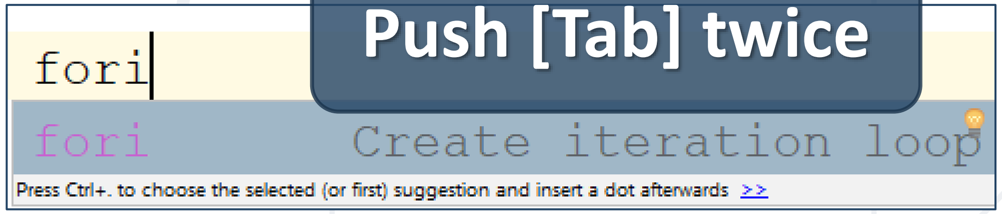
The bracket is again on the same line

Example: Divisible by 3

- Print the numbers from 1 to 100, that are divisible by 3

```
for (int i = 3; i <= 100; i += 3) {
  System.out.println(i);
}
```

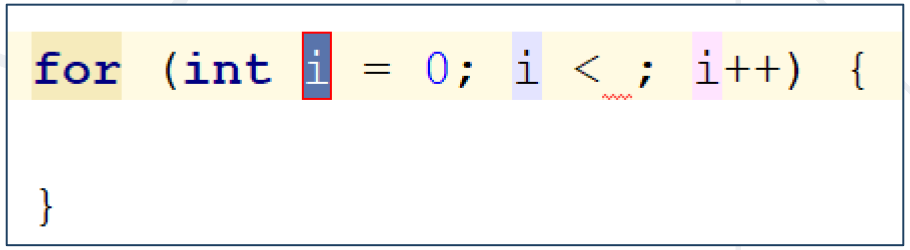
- You can use "**fori**" live template in Intellij



```
fori
fori                Create iteration loop
```

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards >>

Push [Tab] twice



```
for (int i = 0; i < ; i++) {

}
```

Problem: Sum of Odd Numbers

- Write a program to print the first **n** odd numbers and their sum

```
5
```

```
1
3
5
7
9
Sum: 25
```

```
3
```

```
1
3
5
Sum: 9
```

Solution: Sum of Odd Numbers

```java
int n = Integer.parseInt(sc.nextLine());
int sum = 0;

for (int i = 1; i <= n; i++) {
  System.out.println(2 * i - 1);
  sum += 2 * i - 1;
}
System.out.printf("Sum: %d", sum);
```

While Loops

Iterations While a Condition is True

While Loops

- Executes commands while the condition is true:

Initial value

Condition

Loop body

```
int n = 1;
while (n <= 10) {
  System.out.println(n);
  n++;
}
```

Increment the counter

Problem: Multiplication Table

- Print a table holding number*1, number*2, …, number*10

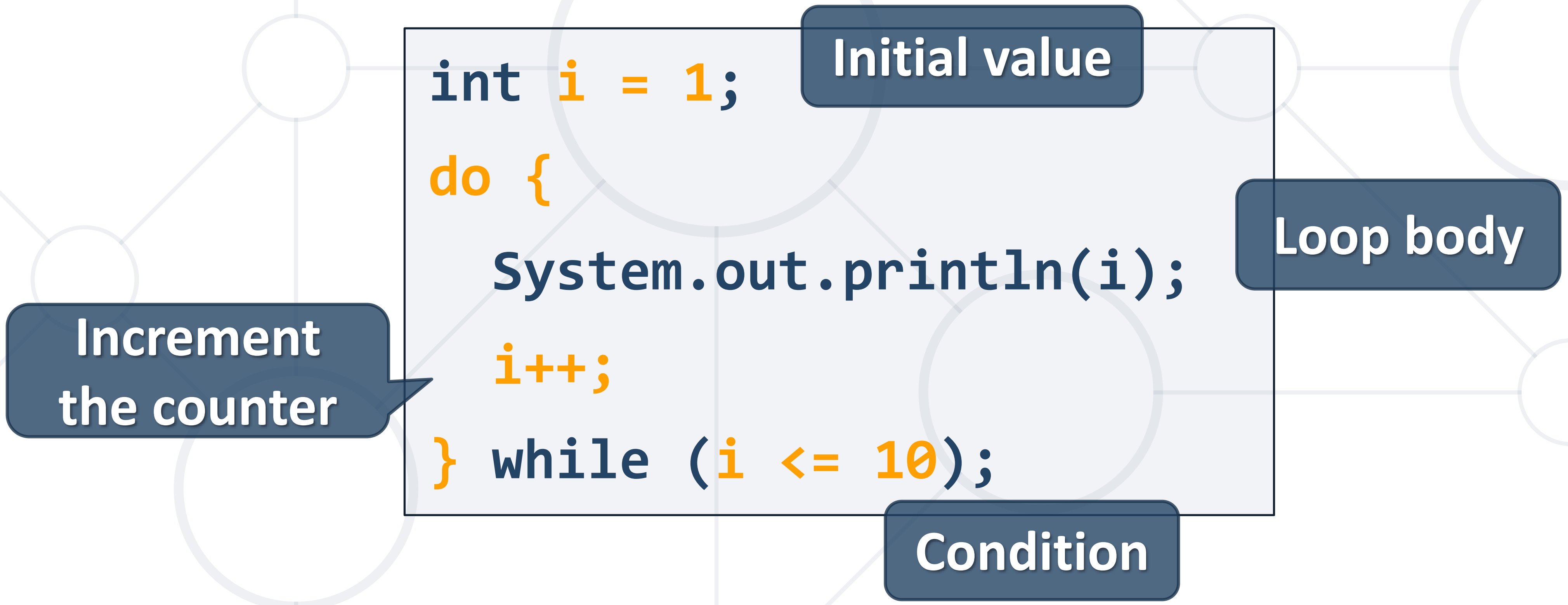
```
int number = Integer.parseInt(sc.nextLine());
int times = 1;
while (times <= 10) {
  System.out.printf("%d X %d = %d%n",
                         number, times, number * times);
  times++;
}
```

Do…While Loop

Execute a Piece of Code One or More Times

Do … While Loop

- Similar to the **while** loop, but always executes at least once:



```java
int i = 1;
do {
  System.out.println(i);
  i++;
} while (i <= 10);
```

Problem: Multiplication Table 2.0

- Upgrade your program and take the initial times from the console

```
int number = Integer.parseInt(sc.nextLine());
int times = Integer.parseInt(sc.nextLine());
do {
  System.out.printf("%d X %d = %d%n",
                number, times, number * times);
  times++;
} while (times <= 10);
```

Check your solution here: https://judge.softuni.bg/Contests/1190/

Debugging the Code

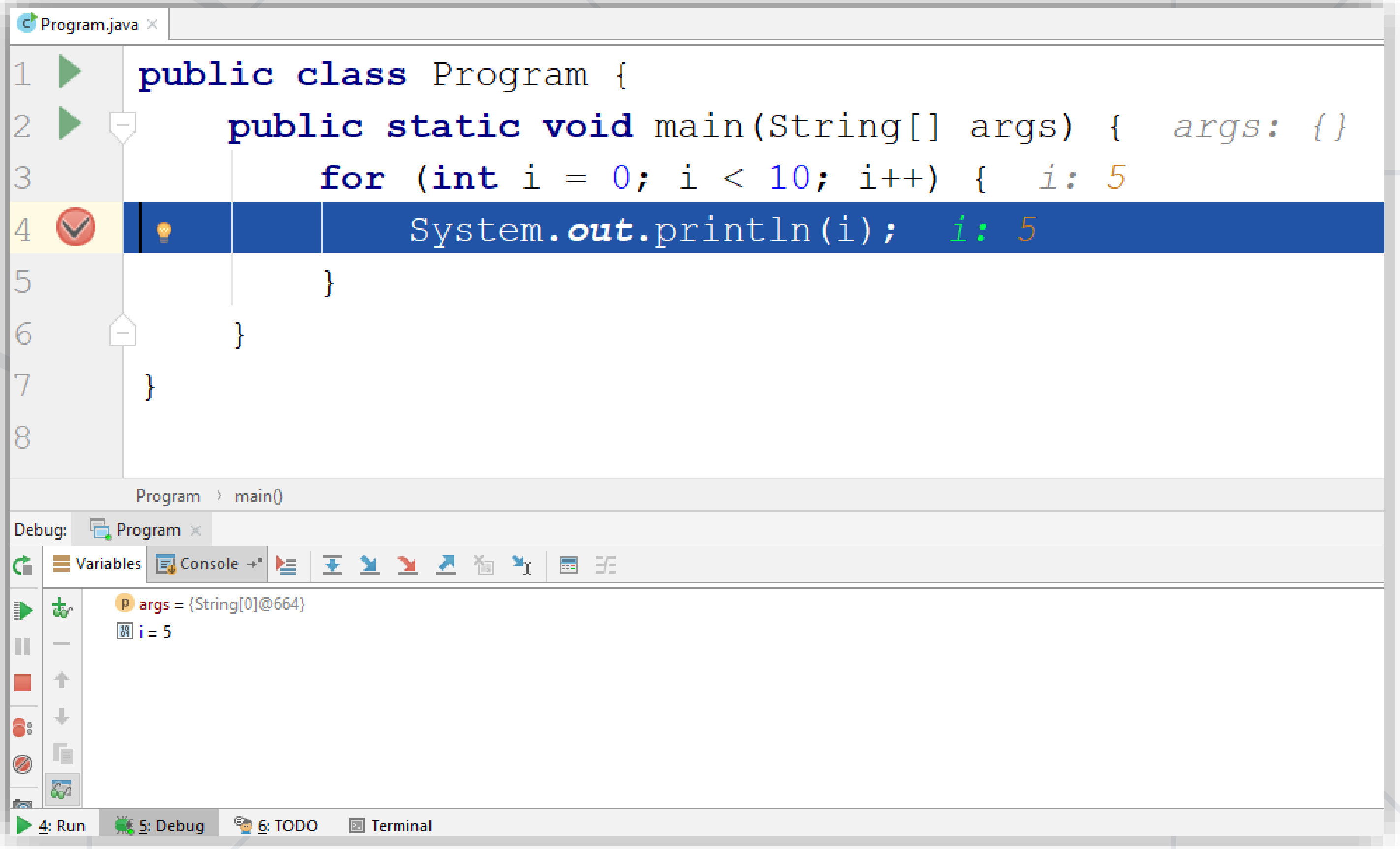
Using the InteliJ Debugger

Debugging the Code

- The process of **debugging application** includes:
  - Spotting an error
  - Finding the lines of code that cause the error
  - Fixing the error in the code
  - Testing to check if the error is gone and no new errors are introduced
- Iterative and continuous process

Debugging in Intellij

- Intellij has a built-in **debugger**
- It provides:
  - **Breakpoints**
  - Ability to **trace** the code execution
  - Ability to **inspect** variables at runtime

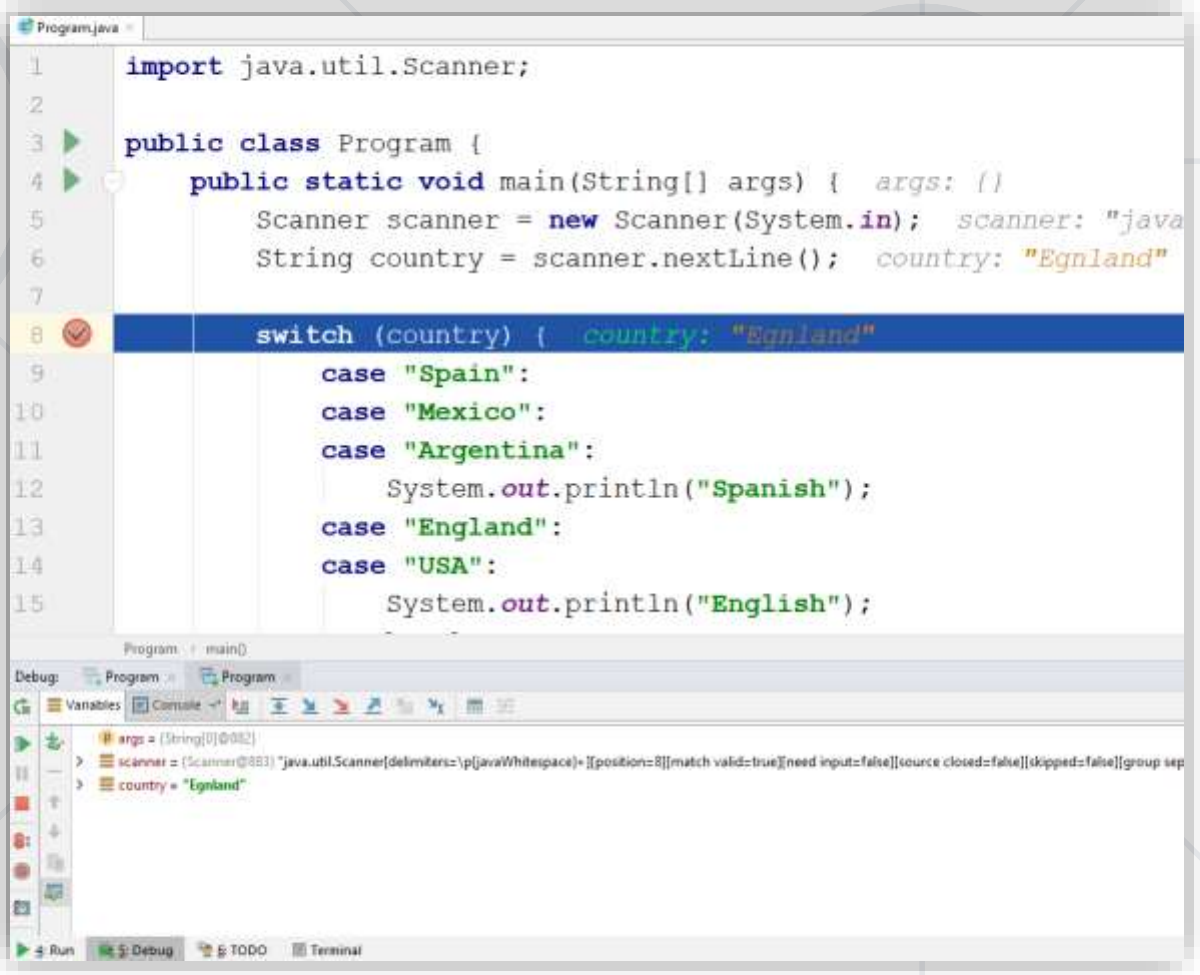


```
public class Program {
    public static void main(String[] args) {  args: {}
        for (int i = 0; i < 10; i++) {  i: 5
            System.out.println(i);  i: 5
        }
    }
}
```

```
args = {String[0]@664}
i = 5
```

Using the Debugger in Intellij

- Start without Debugger: **[Ctrl+Shift+F10]**
- Toggle a breakpoint: **[Ctrl+F8]**
- Start with the Debugger: **[Alt+Shift+F9]**
- Trace the program: **[F8]**
- Conditional breakpoints



Program.java

```
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {   args: {}
        Scanner scanner = new Scanner(System.in);   scanner: "java
        String country = scanner.nextLine();   country: "Egnland"

        switch (country) {   country: "Egnland"
            case "Spain":
            case "Mexico":
            case "Argentina":
                System.out.println("Spanish");
            case "England":
            case "USA":
                System.out.println("English");
```

```
args = {String[0]@882}
scanner = {Scanner@883} "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=8][match valid=true][need input=false][source closed=false][skipped=false][group sep
country = "Egnland"
```

Problem: Find and Fix the Bugs in the Code

- A program aims to print the first **n** odd numbers and their sum

```
Scanner sc = new Scanner(System.in);
int n = Integer.parseInt(sc.nextLine());
int sum = 1;
for (int i = 0; i <= n; i++) {
  System.out.print(2 * i + 1);
  sum += 2 * i;
}
System.out.printf("Sum: %d%n", sum);
```

10

Summary

- Declaring **Variables**
- **Reading** from / **Printing** to the **Console**
- **Conditional Statements** allow implementing programming logic
- **Loops** repeat code block multiple times
- Using the debugger