Java Basics

2020 Spring: AP Computer Science A

December 26th, 2019

Today

- **General Concepts**
- **Java Program Structure**
- **Java Strings**
  - Strings and operation
  - `println()` and `print()`
  - Escape sequences
- **Comments**
- **Java Data Types**
  - Declaration and assignment
  - Representation of numbers
- **Java Operators**
- **Interactive Programs with `Scanner` class**

Programming – General Concepts

- **Program**
  - A **set of instructions** to be carried out by a computer
- **Programming**
  - Creating an ordered set of instructions to solve a problem with c computer
- **Programming language**
  - A systematic set of rules used to describe computations in a format that is editable by humans
  - Ex) Java, C++, Python ...

Programming – General Concepts

- **Syntax**
  - Set of *legal structures and commands* that can be used in a language
  - **Every basic Java statement ends with a semicolon ;**
  - If you violate this, you will get...

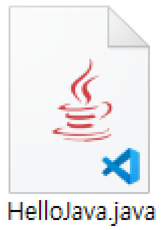
- **Syntax Error (Compile Error)**
  - A problem in the structure of a program that causes compilation failure
    - Missing semicolon
    - Mismatching { } braces
    - Illegal variable names
    - **Class name and file names do not match**
    - ...
  - When error occurs, read the error messages carefully!

Programming – General Concepts

1. *Write it*

- **Code** or **source code**: the set of instructions in a program

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

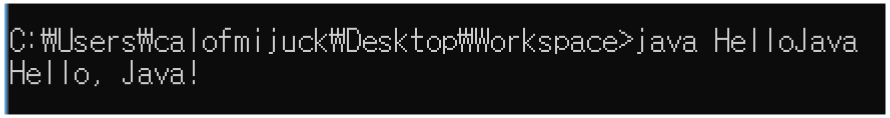


2. *Compile it*

- **compile**: translate a program from one language to another

3. *Execute it*

- The messages printed to the user by a program
- **console**: Text box where the program's output is printed



```
C:₩Users₩calofmijuck₩Desktop₩Workspace>java HelloJava
Hello, Java!
```

Java Program Structure

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

- **class:** a program
  - Class name **must** equal the file name!
- **method:** a named group of statements
- **statement:** a command to be executed

- ***Statements inside main will be executed!***

Java Strings

- String: ***a sequence of characters***
  - Starts and ends with a " (quote) character
  - The quotes do not appear in the output
  - Examples:
    - `"hello"`
    - `"This is a string. It's very long!"`
  - May not span multiple lines
  - May not contain a " character

- **String concatenation**
  - Use + between two strings to make a longer string
    - `"hello, "` + `"world"` is `"hello, world"`

Java Output

- **`System.out.println()`**
  - Prints a line of output on the *console*
  - Always prints new line at the end

- **Two ways to use `System.out.println()`**
  - `System.out.println("message");`
    - Prints the given string (`"message"`) as output
  - `System.out.println();`
    - Prints a blank line of output

- **`System.out.print()`**
  - Prints the given string *without* new line

Exercise

- **Print the following text using Java**

```
Welcome to Java class!
We are learning how to use println()!
```

- **Print the following shape using Java**

```
   *
  ***
 *****
*******
  ***
  ***
```

Escape Sequences

- **Escape Sequence**
  - A special sequence of characters used to represent special characters in a string

`\t`	tab character
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

- **Example:**
  - `System.out.println("\\hello\nhow\tare \"you\"?\\\\");`
  - Output:

```
\hello
how     are "you"?\\
```

Exercise

- **Write a `println` statement to produce this output**
  - All blanks are spaces

```
/ \ // \\ /// \\\
```

- **Use a <u>single</u> `println` statement to produce this output**
  - All blanks are tabs

```
a	b	c
"\''"
Escape
sequences
```

Java Comments

- Comment
  - A note written in source code by the programmer to describe or clarify the code
  - Comments are ignored when your program runs

- **Examples**
  - `// This is a one-line comment`

  -
    ```
    /* This is a
      multi-line comment */
    ```

- **Comments are useful for:**
  - Explaining complex pieces of code or complex programs
  - Multiple programmers working together

Data Types

- **Type**: A category or set of data values
  - Used to represent real-world objects
  - Constrains the operations that can be performed on data
  - Java programmers must specify types
  - Ex) Integers, real numbers, character, string ...

- **Primitive Types:** Built-in types

`int`	Integers (2, -26, 3000)
`double`	Real numbers (3.1, -0.25, 0.001)
`boolean`	logical values (true, false)
`char`	Single characters ('a', 'b', 'c')

  - 4 more: `byte, short, long, float`

Exercise

- **Write the following code and check its output**
  - You can ignore comments

```
public class TypeExample {
    public static void main(String[] args) {
        System.out.println(-1); // -1 (int)
        System.out.println(3.1415); // 3.1415 (double)
        System.out.println('a'); // a (char)
        System.out.println(true); // true (boolean)
    }
}
```

Variables

- ***We want to use these data for computation***
  - *Can we **store** data ?*
  - *Can we perform **operations** on them ?*

- **Variable:** A piece of computer memory that is given a **name** and **type**, and can **store a value**
  - Steps for using a variable

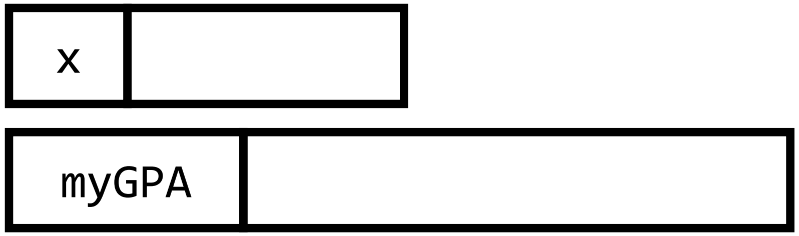
• ***Declare*** it	- State its name and type
• ***Initialize*** it	- Store a value into it
• ***Use*** it	- Print it or use in operation

Variable Declaration

- Variable Declaration
  - Sets aside memory for storing a value
  - **Variables must be declared before usage**

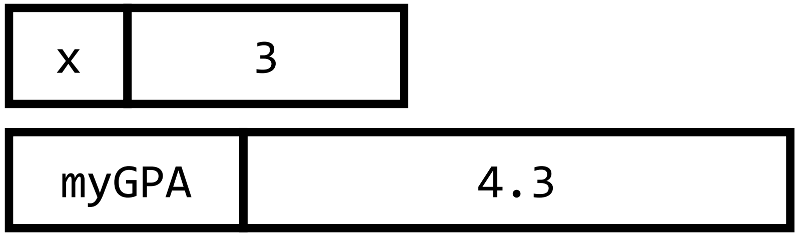
- **Syntax**
  - `type name;`
  - The name is called an ***identifier***

- **Examples**
  - `int x;`
  - `double myGPA;`



Variable Declaration

- **Assignment**
  - Stores a value into a variable
  - **= does not mean equals!**
- **Syntax**
  - **`name = expression;`**
- **Examples**
  - `x = 3;`
  - `myGPA = 3.1 + 1.2;`



Expressions

- **Expression:** A value or operation that computes a value
- **Examples**
  - `1 + 4 * 5                    // 21`
  - `(7 + 2) * 6 / 3              // 18`
  - `42                           // 42`
  - The simplest expression is a *literal value*
  - A complex expression can use operators and parentheses
- **As a program runs, its expressions are *evaluated***
  - `1 + 1` evaluates to 2
  - `System.out.println(3 * 4);  // prints 12`

Using Variables

- Once given a value, a variable can be used in expressions
- You can assign a value more than once

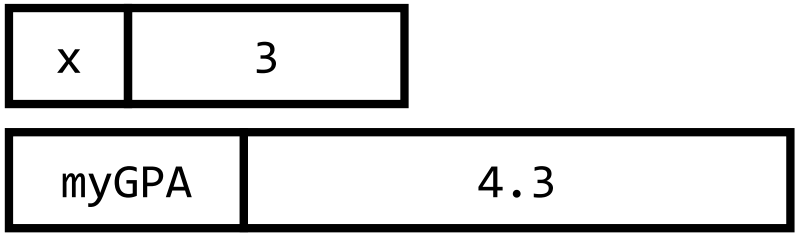
```
public class VariableExample {
    public static void main(String[] args) {
        int x;                               // Declare
        x = 15;                              // Set x to 15
        System.out.print("x is ");
        System.out.println(x);               // x is 15
        System.out.println("x is " + x);
        System.out.println(2 * x - 1);   // 29

        x = 4 + 7;                           // x is now 11
        System.out.println(x);               // 11
    }
}
```

Variable Declaration/Assignment

- A variable can be declared and initialized in one statement.

- Syntax
  - `type name = value;`

- Examples
  - `int x = 3;`
  - `double myGPA = 4.3;`

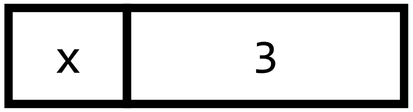


Assignment

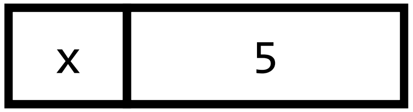
- **= is called an assignment operator**
  - Does not mean equals!
  - Means: *"Store the value at right in variable at left"*

- **The right-side expression is evaluated first, and the result is stored in the variable at left**

- **Example**
  - `int x = 3;`



  - `x = x + 2;`



  - x + 2 is evaluated and stored in x

Assignment and Compile Errors

- **A variable can only store a value of its own type**
  - `int x = 2.5;        // error: incompatible types`
  - `double x = 2;       // OK. 2 is a real number`

- **A variable can't be used until it is assigned a value**
  - `int x;`
  - `System.out.println(x);        // error: x might not have been initialized`

- **You may not declare the same variable twice**
  - `int x;`
  - `int x;               // error: variable x is already defined`

Identifiers and Keywords

- **Identifier**: A name given to an item in your program
  - Must start with a letter or _ or $
  - Subsequent characters can be any of those or a number
  - Legal identifiers
    - `_myName, TheCure, ANSWER_IS_42, $bling$`
  - Illegal identifiers
    - `me+u, 49ers, side-swipe, Ph.D's`
- **Camelcase Convention**
  - When naming a variable with multiple words, capitalize each word except for the first
  - Ex) `myVariableName, longVariableName, totalSum`
- **Keyword**: An identifier that you cannot use because it already has a reserved meaning in Java
  - **`int, double, boolean`** …

Representation of Numbers

- Digital devices have two stable states, 0 and 1
- The binary number system has two digits, 0 and 1
- **A single digit (0 or 1) is called a *bit*, short for binary digit**
- **1 byte = 8 bits**

- **Decimal Integers (Base 10)**
  - Uses ten digits (0 ~ 9)
  - Position values are powers of 10
  - n decimal digits can represent 10^n unique values

- **Binary Integers (Base 2)**
  - Uses two digits (0, 1)
  - Position values are powers of 2
  - n binary digits can represent 2^n unique values

Representation of Numbers

- **How to count in binary**

	128	64	32	16	8	4	2	1
117	0	1	1	1	0	1	0	1

- **Java `int` uses 4 bytes (32 bits)**
  - 1 bit is used for sign
  - Stores numbers from $-2^{31} \sim 2^{31} - 1$
  - Generally, n bit integer can store $-2^{n-1} \sim 2^{n-1} - 1$

Representation of Numbers

- **Java double**

sign	***exponent***	***mantissa***
1 bit	*11 bits*	*52 bits*

  - Uses scientific notation
  - $(-1)^{sign} * mantissa\ * 2^{exponent}$

- **Mantissa has *finitely* many digits**
  - Causes *round-off errors*
  - Somewhat different from real numbers

Java Operators

- **Operator:** Computation that combines multiple values or expressions
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Increment and Decrement Operators

Arithmetic Operators

- **Used for calculation involving numbers**

Operator	Meaning	Example
`+`	Addition	`3 + x`
`-`	Subtraction	`p – q`
`*`	Multiplication	`6 * i`
`/`	Division	`10 / 4`
`%`	Remainder (mod)	`11 % 8`

- **Can be applied to numerical types**
  - **`int, double,`** `(byte, short, long, float)`

- **+** *can also be used to **concatenate data with strings***

Integer Arithmetic

- **Integer division returns the *quotient*!**
  - Ex. `14 / 4` is 3, not `3.5`
  - Division by 0 causes an error
- **% operator computes the remainder from integer division**
  - Ex. `14 % 4` is 2
  - Check if x is odd: `x % 2`
  - Obtain last digit of x : `x % 10`
- ***Subtle when handling negative integers***
  - `-4 / 3` is `-1`
  - `-5 % 3` is `-2`

Real Number Arithmetic

- **Examples:** `6.022, -42.0, 3.1415`
  - Placing `.0` or `.` after an integer makes it a **`double`**

- **/ produces an *exact answer***
  - `15.0 / 2.0` is `7.5`

- **When `int` and `double` are mixed, the result is a `double`**
  - `4.2 * 3` is `12.6`
  - `7.2 / 3` is `2.4`

Exercise

- **Calculate the answer of the following expression**
  - `123 + 456 * 789 / 3 % 2`

- **Follow the steps.**
  - Declare a variable `x` and assign `30`
  - Declare a variable `y` and assign `15`
  - Print `x + y`, `x - y`, `x * y`, `x / y`, `x % y`, respectively on each line

Relational Operators

- **Determine relations between values**

Operator	Meaning	Example
`==`	Equal to	`x == 10`
`!=`	Not equal to	`10 != 11`
`>`	Greater than	`3 > x`
`<`	Less than	`2 < 10.0`
`>=`	Greater than or equal to	`3.14 >= 3.1`
`<=`	Less than or equal to	`2.718 <= e`

- **Relational operators are used in *boolean expressions***
  - Boolean expressions will evaluate to **`true`** or **`false`**
  - Ex. 2 > 3 will evaluate to **`false`**

Logical Operators

- Logical operators are applied to *boolean expressions* to form *compound boolean expression* that evaluate to `true` or `false`

Operator	Meaning	Example
!	Logical NOT	`!x`
&&	Logical AND	`3 < x && x < 5`
||	Logical OR	`x > 5 || x < -2`

- Truth Tables

!	
T	F
F	T

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

Assignment Operators

- **Provides compact form**

Operator	Example	Meaning
`=`	`x = 2`	Simple assignment
`+=`	`x += 4`	`x = x + 4`
`-=`	`y -= 6`	`y = y - 6`
`*=`	`p *= 5`	`p = p * 5`
`/=`	`n /= 10`	`n = n / 10`
`%=`	`n %= 10`	`n = n % 10`

- **Chaining assignment is allowed, with evaluation from right to left**
  - `next = prev = sum = 0;`
  - Initializes `sum` to `0`, `prev` to `sum`, `next` to `prev`

Increment/Decrement Operators

Operator		Example
`++`	Pre-increment	`++i`
`++`	Post-increment	`i++`
`--`	Pre-decrement	`--i`
`--`	Post-decrement	`i--`

- **Increase or decrease the value in variable by 1**
  - Pre-in/decrement - Calculated ***on*** evaluation
  - Post-in/decrement - Calculated ***after*** evaluation

- **Example**

```
int i = 5, j = 3;
System.out.println(++i);   // prints 6
System.out.println(j++);   // prints 3, j is incremented to 4
System.out.println(j);     // prints 4
```

Operator Precedence

- **Operator precedence: Order of operator evaluation in expression**
  1. `!, ++, --`
  2. `*, /, %`
  3. `+, -`
  4. `<, >, <=, >=`
  5. `==, !=`
  6. `&&`
  7. `||`
  8. `=, +=, -=, *=, /=, %=`
- **Parentheses are always evaluated first**
- **Associativity: Order of evaluation on operators with same precedence**
  - Right to left for 1, 8
  - Left to right otherwise

Exercise

- **Guess the output/value without running the code!**
  - `5 + 3 < 6 - 1`
  - `"asdf" + 1 + 2`
  - `1 + 2 + "asdf"`
  - `!(3 >= 4) && (4 != 3)`

  - `int i = 5;`
  - `int x = i++;`
  - `x > i;`
  - `x += i;`

Interactive Programs

- **An *interactive program* reads input from the console**
  - While the program runs, it asks the user to type input
  - The input typed by the user is stored in variables in the code
- **Interactive programs have more interesting behavior!**
- **Use `Scanner` class to receive input from user!**

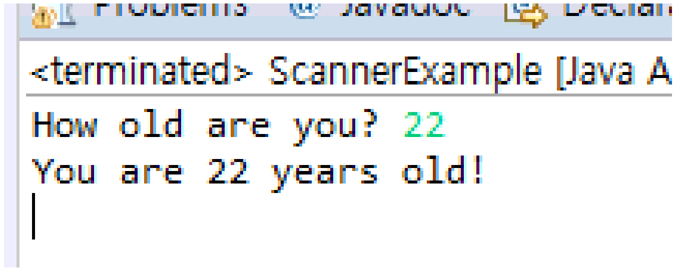
Scanner Class

- **Scanner class is found in the java.util package**
  - Write the following line at the top of your source code
  - `import java.util.Scanner;`

- **Declaring a Scanner**
  - `Scanner name = new Scanner(System.in);`

- **Example**
  - `Scanner sc = new Scanner(System.in);`

Scanner Example



```
<terminated> ScannerExample [Java A
How old are you? 22
You are 22 years old!
```

- **Run the following code**

```java
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("How old are you? ");
        int x = sc.nextInt();
        System.out.println("You are " + x + " years old!");
    }
}
```

- **The console waits for the user to type the input and press Enter**
- **The value typed by the user is returned**

- **prompt:** A message telling the user what input to type

Scanner Usage

- `Scanner sc = new Scanner(System.in);`

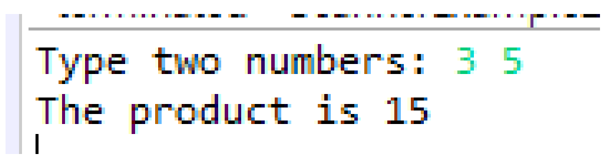
Method	Description
`sc.nextInt()`	Reads an **`int`** from the user
`sc.nextDouble()`	Reads a **`double`** from the user
`sc.next()`	Reads a *one-word* **`string`** from the user

- **Usage**

```
int x = sc.nextInt();          // reads int and stores it into x
double y = sc.nextDouble();    // reads double and stores it into y
String s = sc.next();          // reads string and stores it into s
```

Scanner Example 2

- **Run the following code**



```
Type two numbers: 3 5
The product is 15
```

```java
import java.util.Scanner;

public class ScannerExample2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Type two numbers: ");
        int x = sc.nextInt();
        int y = sc.nextInt();
        int product = x * y;
        System.out.println("The product is " + product);
    }
}
```

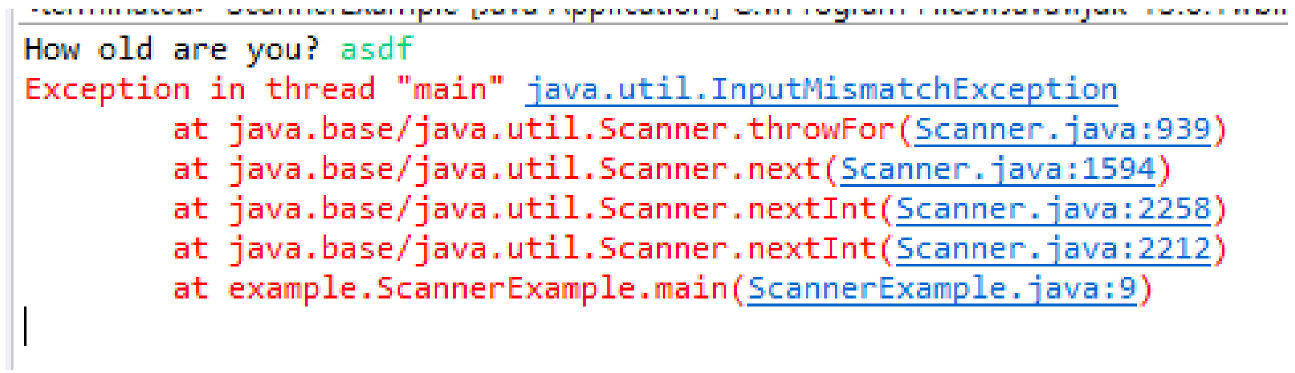
- **The Scanner can read multiple values from one line**
  - They values must be separated ...

Scanner Input Tokens

- **token:** A unit of user input, as read by the Scanner
  - Tokens are separated by ***whitespace*** (spaces, tabs, new lines)
  - How many tokens are there in the following line?

```
23  John Smith   42.0  "Hello world"  $2.50  "   19"
```

- **When a token is not the type you ask for, the program crashes**
  - Refer to `ScannerExample`

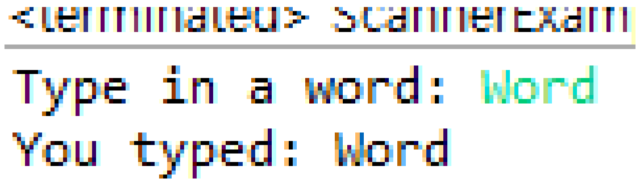


```
How old are you? asdf
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at example.ScannerExample.main(ScannerExample.java:9)
```

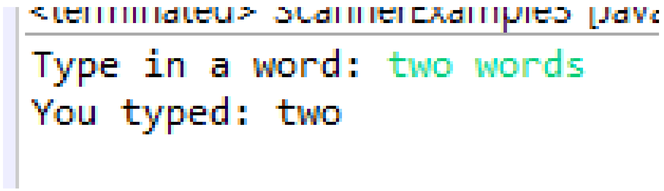
Strings as User Input

- **Scanner's next() method reads a token as a String**
- **String declaration**
  - `String str = "This is a string";`
- **Example**

```java
import java.util.Scanner;

public class ScannerExample3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Type in a word: ");
        String str = sc.next();
        System.out.println("You typed: " + str);
    }
}
```



```
Type in a word: Word
You typed: Word
```

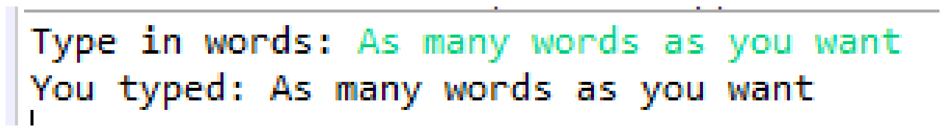


```
Type in a word: two words
You typed: two
```

Strings as User Input

- **To read multiple words in a line, use `nextLine()`**

```
import java.util.Scanner;

public class ScannerExample4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Type in words: ");
        String line = sc.nextLine();
        System.out.println("You typed: " + line);
    }
}
```



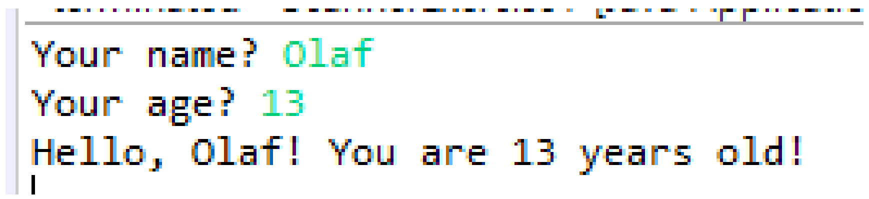
```
Type in words: As many words as you want
You typed: As many words as you want
```

- **Do not use `nextLine` mixed with `next?` methods**

Exercise

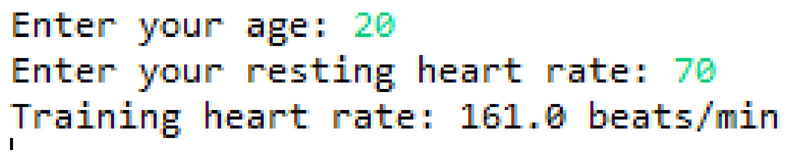
- User input is shown in green text

- *You are given a single word and an integer as input, separated by new line.*
  Write a program that produces the following output



```
Your name? Olaf
Your age? 13
Hello, Olaf! You are 13 years old!
```

- *You are given two integer as input, separated by new line*
  - a*: age,* r*: resting heart rate*
  - *Training heart rate =* $0.7\ (220 - a) + 0.3\ r$
  - *Produce the following ouput*



```
Enter your age: 20
Enter your resting heart rate: 70
Training heart rate: 161.0 beats/min
```

BOJ

- **Register on** https://acmicpc.net
- We will solve a lot of problems as homework!