Programming 1

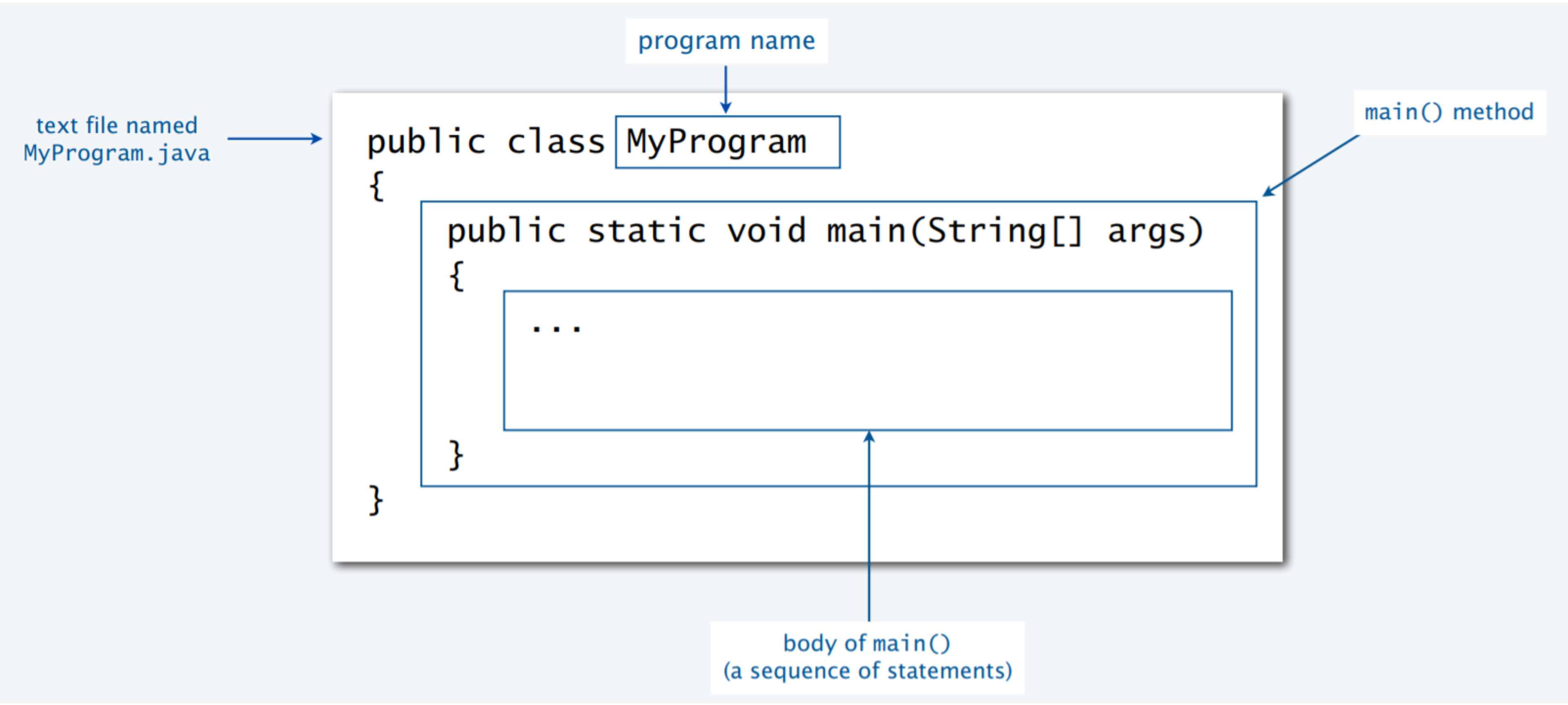
Lecture 2 – Basic Java syntax

Variables & Data types 1

Basic decision control

Faculty of Information Technology
Hanoi University

Structure of simple Java programs



```
public class MyProgram
{
   public static void main(String[] args)
   {
      ...
   }
}
```

Image Credit: **R. Sedgewick**

Single-line comments

- Comments are used to
  - Explain code
  - Prevent code execution (good for code testing)

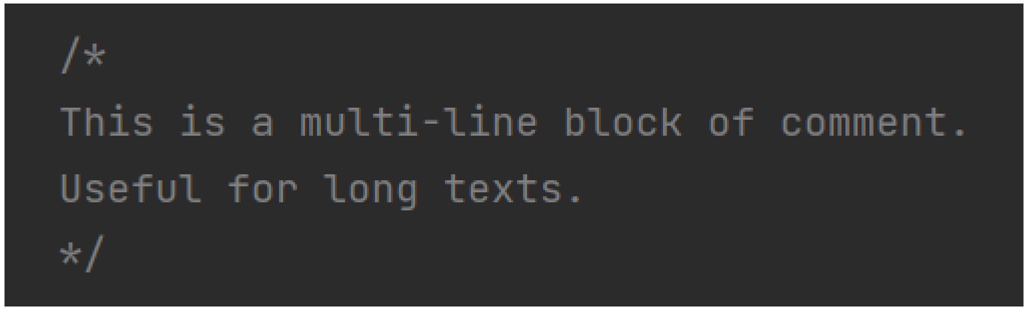
```
// some explanation
System.out.println("I love programming!");
```

```
int x = 5;
// x = x + 3;
System.out.println(x);
```

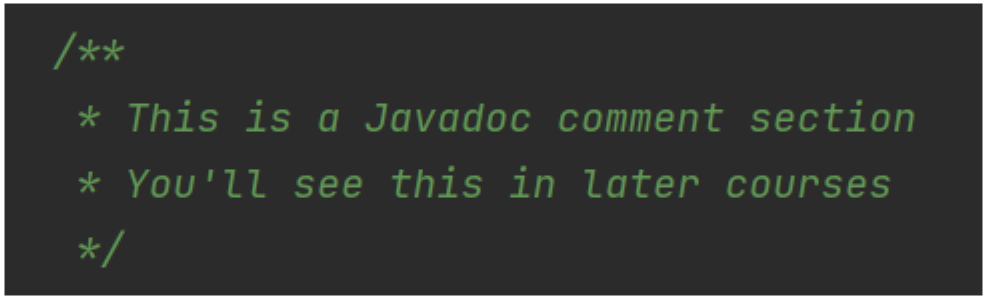
```
int x = 5;
x += 3; // increase x
```

Multi-line comments

- Two types:
  - **Block comment** starts with /* and ends with */
  - **Javadoc comment** starts with /**, ends with */ and each line begins with a *.



```
/*
This is a multi-line block of comment.
Useful for long texts.
*/
```



```
/**
 * This is a Javadoc comment section
 * You'll see this in later courses
 */
```

Three versions of the same program.

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

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *************************************************************************/

public class HelloWorld {

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

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

Fonts, color, comments, and extra space are not relevant in Java

What is a variable?

A named value

```
int age = 19;
```

I use the name **age** to call the number **19**

So that when I say:

I am **age** years old!

You would understand that I was 19 years old.

```
String name = "Quan";
```

Welcome, **name**!

would be equivalent to:

Welcome, Quan!

What is a literal?

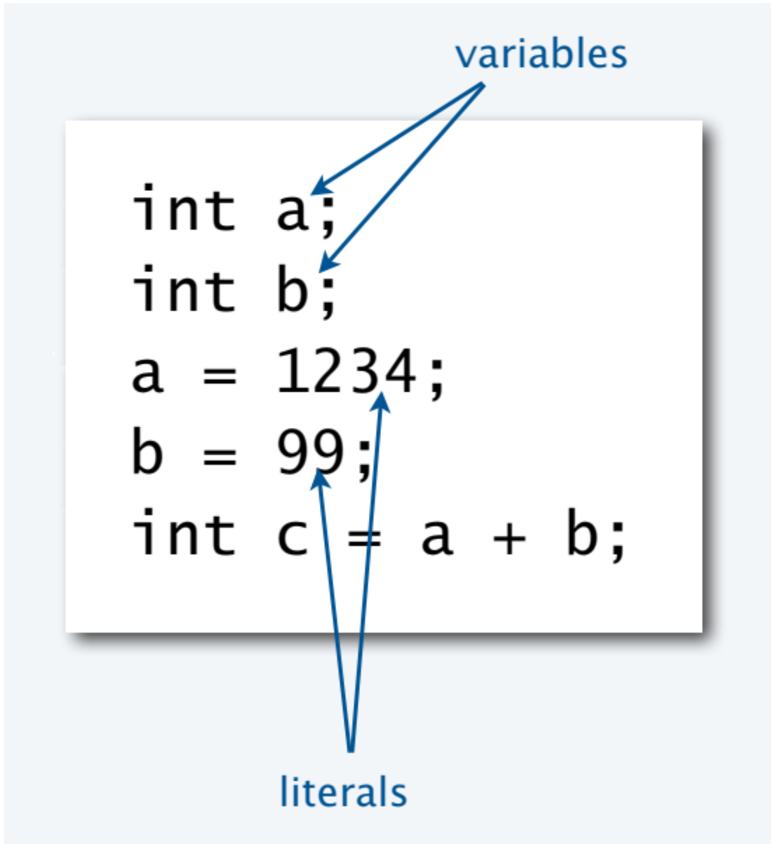
A programming-language representation of a value

```
int age = 19;
```

```
19 is an integer
literal.
```

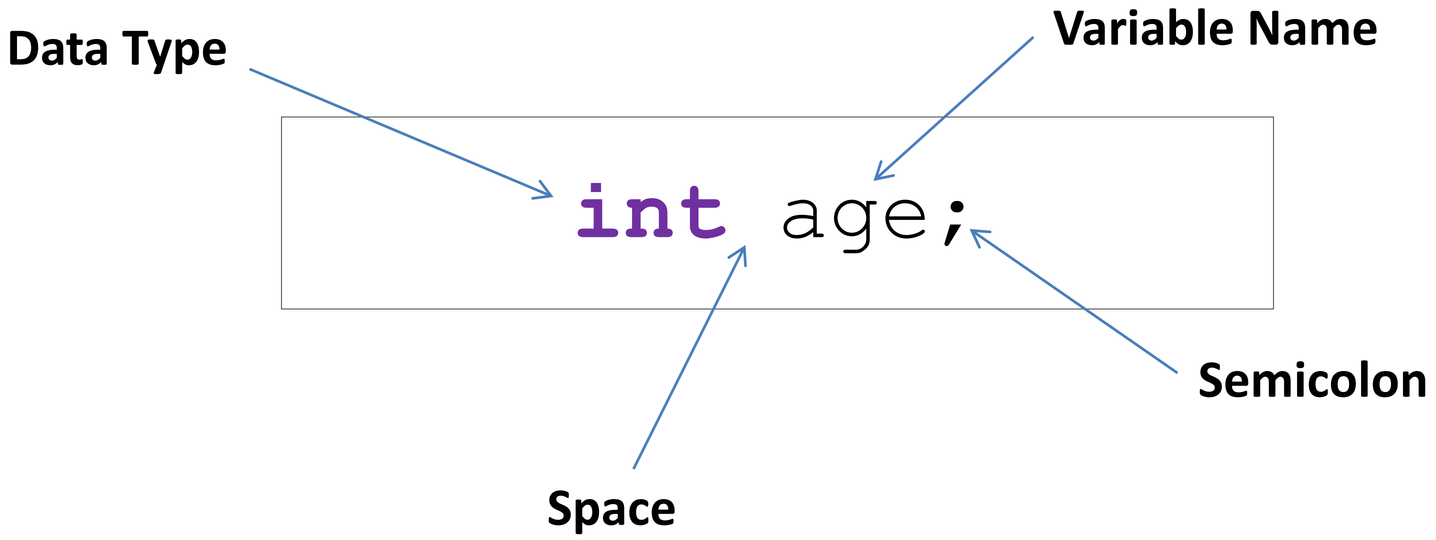
```
String name = "Quan";
```

```
"Quan" is a String literal.
```



```
int a;
int b;
a = 1234;
b = 99;
int c = a + b;
```

Declaring a variable



```
int age;
```

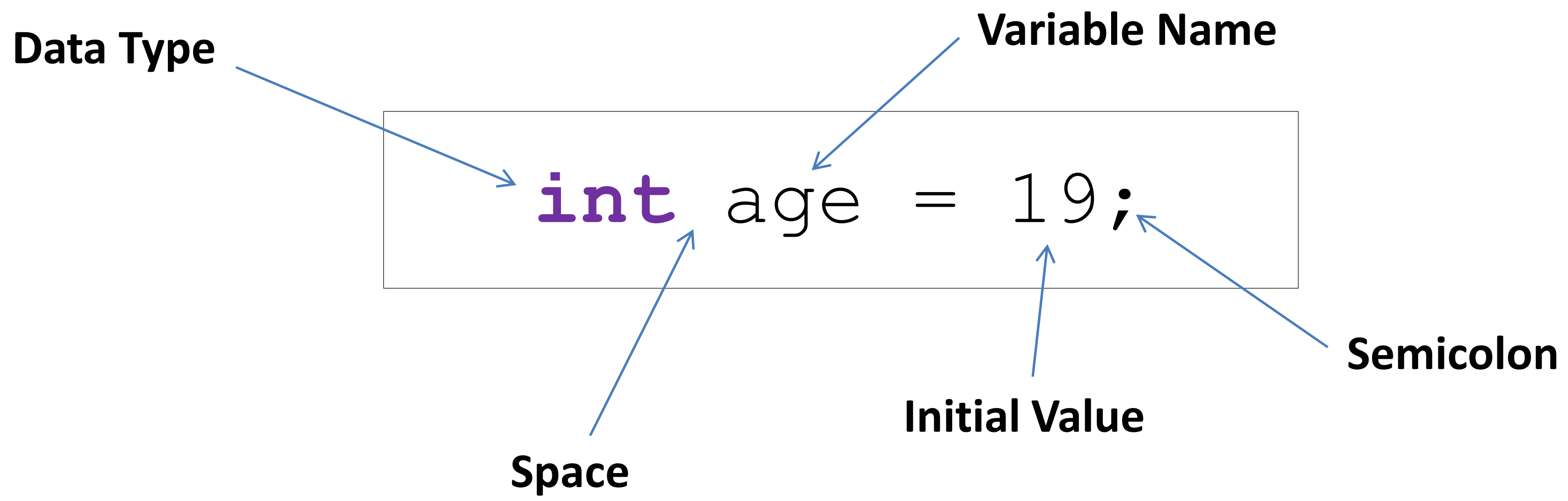
A declaration statement associates a variable with a type.

You cannot re-declare existing variables!

Basic (primitive) data types

Type	Description	Default	Size	Value range
boolean	True or False	false	1 bit	false, true
byte	Very small integers	0	8 bits	(-2^{8}) .. $(2^{8} - 1)$ -128 .. 127
short	Small integers	0	16 bits	(-2^{15}) .. $(2^{15} - 1)$
int	Integers	0	32 bits	(-2^{31}) .. $(2^{31} - 1)$
long	Bigger integers	0	64 bits	(-2^{63}) .. $(2^{63} - 1)$
float	Real numbers	0.0	32 bits	1.424×10^{-45} .. 3.4028×10^{38}
double	More precise real numbers	0.0	64 bits	4.9406×10^{-324} .. 1.7977×10^{308}
char	UTF-16 character	\u0000	16 bits	\u0000 .. \uFFFF

Declare & initialize a variable



```
int age = 19;
```

Initialize: give the variable an initial value.

Notes on Java variables

- A variable can be declared only once in a simple program
- Variable name is a kind of Java Identifier and follows Java identifier rules
- Identifiers are names given to things in Java (variables, classes, methods, etc.)

Java Identifier Rules

- An *identifier* can contain letters (a-z, A-Z), digits (0-9), underscores (_) and dollar ($) signs.
- The first character of an *identifier* cannot be a digit.
- Some valid identifiers: abc, product_manager, _body, bin2dec, $address, xtr3$$
- Some invalid identifiers:
  - my-age (the hyphen “-” is not allowed)
  - Hello world (contains a space)
  - 1000words (starts with a digit)

Expressions

- An expression is any piece of code which evaluates to a single value.
- Abstract and hard-to-understand, isn't it?
- Examples:

```
15 + 3
```

```
A math expression which
valuates to 18
```

```
15
```

```
A single value is the most
basic expression
```

More Expressions

`6 * (5 + 3)`	evaluates to **48**
`6 * 5 + 3`	evaluates to **33**
`a`	similar to single-valued expressions
`a + 1`	evaluates to **a's value + 1**
`6 > 5`	A boolean expression which evaluates to **true**
`6 == 5`	evaluates to **false**

Math Operators

`5 + 3`	addition, evaluates to **8**
`5 * 3`	multiplication, evaluates to **15**
`5 - 3`	substraction, evaluates to **2**
`5 / 3`	integer division, evaluates to **1**
`5.0 / 3.0`	float division, evaluates to **1.6667**
`5 % 3`	Modulo (mod), evaluates to **2** (taking the remainder of division)

Statements

- A line of code that commits something. A statement ends with ; in Java.
- Examples:

```
int age = 19;
```

```
This statement declares and
initializes a variable
```

```
age = 19 + 1;
```

```
…performs some calculation
```

```
System.out.println("Hi");
```

```
Java says Hi
```

Assignment statements

- We can imply: $2 + 3 \rightarrow 5$
- Can we imply this? $5 \rightarrow 2 + 3$

$$1 + 4 \rightarrow 5$$

$$7 - 2 \rightarrow 5$$

- In math, we can write these equations:

$$5 = 2 + 3$$

$$2 + 3 = 5$$

Assignment statements

- To express $2 + 3 \rightarrow 5$ or $2 + 3 = 5$ in Java:

```
int a;
a = 2 + 3;
```

Assignment Statement

- A statement does (commits) something:
  - ❑ a wasn't 5 before the statement
  - ❑ after the statement, a holds the value of 5
- Assign an initial value to a variable:

```
int a = 2 + 3;
```

Assignment statements

- General form of an assignment statement:

```
<variable> = <expression>;
```

Implication vs Assignment

- In math, we have the left side and want to figure out the right side (you do the calculation).

2 + 3 = ?

then

2 + 3 = 5

- In programming, we have the right side and computer calculates the left side for us.

a = 2 + 3

then

a gets the value of 5

Assignment statements

```
x = x + 1;
```

Confused?

```
int x = 5;
x = x + 1;
```

What will *x*'s value be?

Assignment statements

```
int x = 5;
x = x + 1;
```

means

```
x = 5 + 1;
```

the shorter way (more on this later)

```
x++;
```

Display expression's value

```
public class Example {
      public static void main(String[] args) {
            int a = 15;
            System.out.println(a);
      }
}
```

Output:

```
15
```

Display expression's value

```
public class Example {
        public static void main(String[] args) {
                int a = 15;
                System.out.println(a*3);
        }
}
```

Output:

```
45
```

Display expression's value with text

```
public class Example {
      public static void main(String[] args) {
            int age = 20;
            System.out.println("I am " + age);
      }
}
```

Output:

```
I am 20
```

Display variable's value with text

```java
public class Example {
	public static void main(String[] args) {
		int age = 20;
		System.out.println(age + " is my age!");
	}
}
```

Output:

```
20 is my age!
```

Display variable's value with text

```java
public class Example {
    public static void main(String[] args) {
        int age = 20;
        System.out.println("I am " + age + " years old");
    }
}
```

Output:

```
I am 20 years old
```

Exercise

- Write a program to get the result of 152 × 1132
- Expected results: 172064

Answer

```java
public class Exercise {
	public static void main(String[] args) {
		int a = 152;
		int b = 1132;
		int c = a * b;
		System.out.println(c);
	}
}
```

Equivalent Answer

```
public class Exercise {
      public static void main(String[] args) {
            System.out.println(152 * 1132);
      }
}
```

Exercise

- Write a program to get the result of the following math operation:

$$\frac{5}{2}$$

- Expected results: 2.5

Answer

```
public class Exercise {
        public static void main(String[] args) {
                double a = 5;
                double b = 2;
                double c = a / b;
                System.out.println(c);
        }
}
```

Equivalent Answer

```java
public class Exercise {
        public static void main(String[] args) {
                double a = 5;
                System.out.println(a / 2);
        }
}
```

Non-Equivalent Answer

```
public class Exercise {
      public static void main(String[] args) {
            System.out.println(5 / 2);
      }
}
```

- Actual result: 2
- Reason: integer division is used when both sides are integer.

Another Equivalent Answer

```
public class Exercise {
	public static void main(String[] args) {
		System.out.println(5.0 / 2);
	}
}
```

- **Fix:** write one of the two values as a real number.

Exercise

- Given 34932 seconds, calculate and display the number of hours, minutes and seconds.
- Expected results:

```
9h, 42m, 12s
```

Answer 1

```java
public class Exercise {
    public static void main(String[] args) {
        int a = 34932;
        int s = a % 60;
        a = a / 60;
        int m = a % 60;
        int h = a / 60;
        System.out.println(
            h + "h, " + m + "m, " + s + "s"
        );
    }
}
```

Answer 2

```java
public class Exercise {
    public static void main(String[] args) {
        int s = 34932 % 60;
        int m = 34932 / 60 % 60;
        int h = 34932 / 60 / 60; // divide by 3600
        System.out.println(
            h + "h, " + m + "m, " + s + "s"
        );
    }
}
```

Answer 3

```java
public class Exercise {
    public static void main(String[] args) {
        int h = 34932/3600;
        int m = (34932 % 3600) / 60;
        int s = (34932 % 3600) % 60;
        System.out.println(
            h + "h, " + m + "m, " + s + "s"
        );
    }
}
```

Boolean expressions

- **Boolean** or **truth** values are: True, False
- A Boolean expression evaluates to a Boolean value

```
      int a = 5; int b = 6;
boolean x = true; boolean y = false;
```

Expression	Boolean value	Explanation
`5 == 5`	`true`	
`5 >= 6`	`false`	5 is neither greater than 6 nor equal to 6
`a == a`	`true`	
`b < a`	`false`	6 is not smaller than 5
`a == 5 && b < 10`	`true`	• `&&` means AND • `a == 5` evaluates to `true` • `b < 10` is also `true` • `true` AND `true` $\longrightarrow$ `true`

Boolean expressions

```
    int a = 5; int b = 6;
boolean x = true; boolean y = false;
```

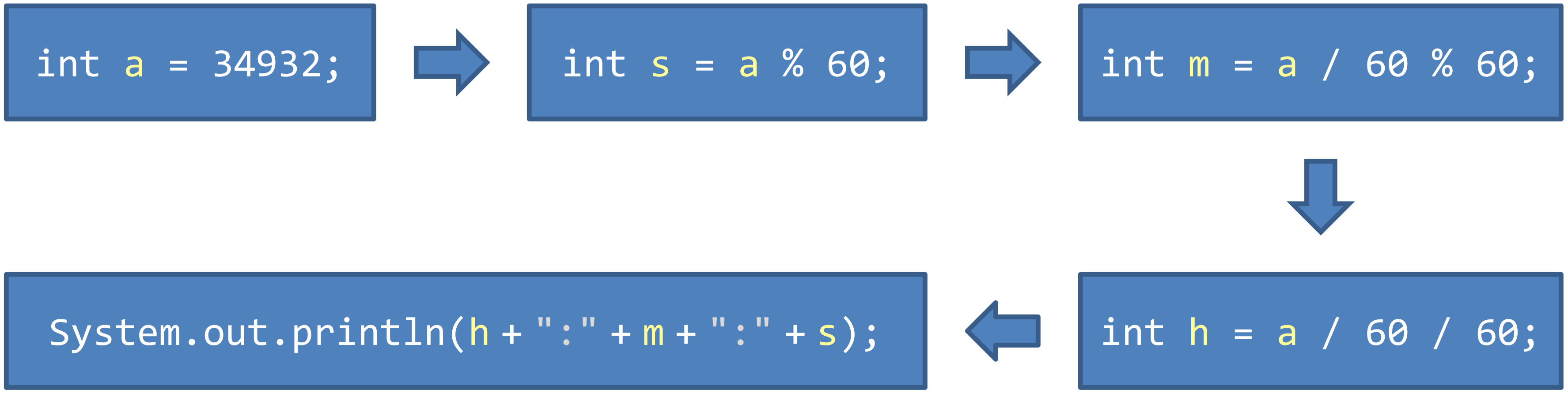
Expression	Boolean value	Explanation
`a == 6 || b == 6`	`true`	• `||` means OR • `a == 6` evaluates to `false` • `b == 6` evaluates to `true` • `false` OR `true` → `true`
`a != 5`	`false`	`!=` means NOT EQUAL but `a` is actually equal to `5` so the expression is `false`
`!(a == 5)`	`false`	it says: "not (`a` equals `5`)" the **!** operator reverses any boolean value that follows (making `true` become `false`)
`x`	`true`	`x` itself is a Boolean value
`x && y`	`false`	the && operator evaluates to `true` only when both values are `true`

Boolean operators

Operator	Meaning	Example
`==`	`true` if both sides are equal	`a == b`
`!=`	`true` if two sides are different	`a != b`
`<`	`true` if the left side is smaller than the right side	`a < b`
`>`	`true` if the left side is greater than the right side	`a > b`
`<=`	smaller or equal to	`b <= a`
`>=`	greater or equal to	`c >= b`
`&&`	AND	`a > b && b > c`
`||`	OR	`a == b || a == c`
`!`	NOT	`!(a < b)`

Execution path

- A program starts at the beginning of `main()` and exits at the end of `main()`.
- The order of statements being executed is called the **execution path**.
- The programs you've seen have **straight** paths.



```
int a = 34932;
int s = a % 60;
int m = a / 60 % 60;
int h = a / 60 / 60;
System.out.println(h + ":" + m + ":" + s);
```

Execution path

- Straight programs do the same thing everytime... not very intelligent.
- An intelligent program would evaluate the situation and choose the most suitable action. For instance:
  - If the user enters incorrect password for 5 times, disable login for 10 minutes.
  - If an Internet user has just searched for "how to write cool software", show him ads about Mr. Quan's programming course.

`if` statement

- Controls the execution of some statements based on a Boolean value.

```
// assume that a and b are entered by user
if (a < 0 || b < 0) {
    System.out.println("Please enter non-negative numbers");
}
```

- The above piece of code shows an error message when the user provides unsuitable inputs and does nothing otherwise.

`if` statement

Boolean expression inside parentheses

Start of block

```
if (a < 0 || b < 0) {
   a = 0;
   b = 0;
   System.out.println("…");
}
```

Statements to be executed if the Boolean expression evaluates to true

End of block

if statement

```
if (a < 0 || b < 0)
   System.out.println("…");
```

Curly braces { } can be omitted if the block contains only one statement

***Note:** a code block only needs curly braces **{ }** if it has multiple statements

`if…else` statement

```
// assume that a and b are entered by user
if (a < 0 || b < 0) {
    System.out.println("Please enter non-negative numbers!");
} else {
    System.out.println("You entered correctly!");
}
```

- The above piece of code shows an error message when the user provides unsuitable inputs and shows a successful message otherwise.

if…else statement

Boolean expression inside parentheses

```
if (a < 0 || b < 0) {
   a = 0;
   b = 0;
   System.out.println("…");
} else {
   System.out.println("ok");
   System.out.println(a + b);
}
```

Statements to be executed if the condition is true

Executed if condition is false

Example

- Write a program to solve

$$ax + b = 0$$

  given a and b

- For `a = 2` and `b = 1`, expected result:

$$x = -0.5$$

Answer

```java
public class LinearEQ {
    public static void main(String[] args) {
        double a = 2, b = 1, x;
        if (a != 0) {
            x = -b/a;
            System.out.println("x = " + x);
        }  else {
            if (b == 0)
                System.out.println("holds for any x");
            else
                System.out.println("no solution");
        }
    }
}
```