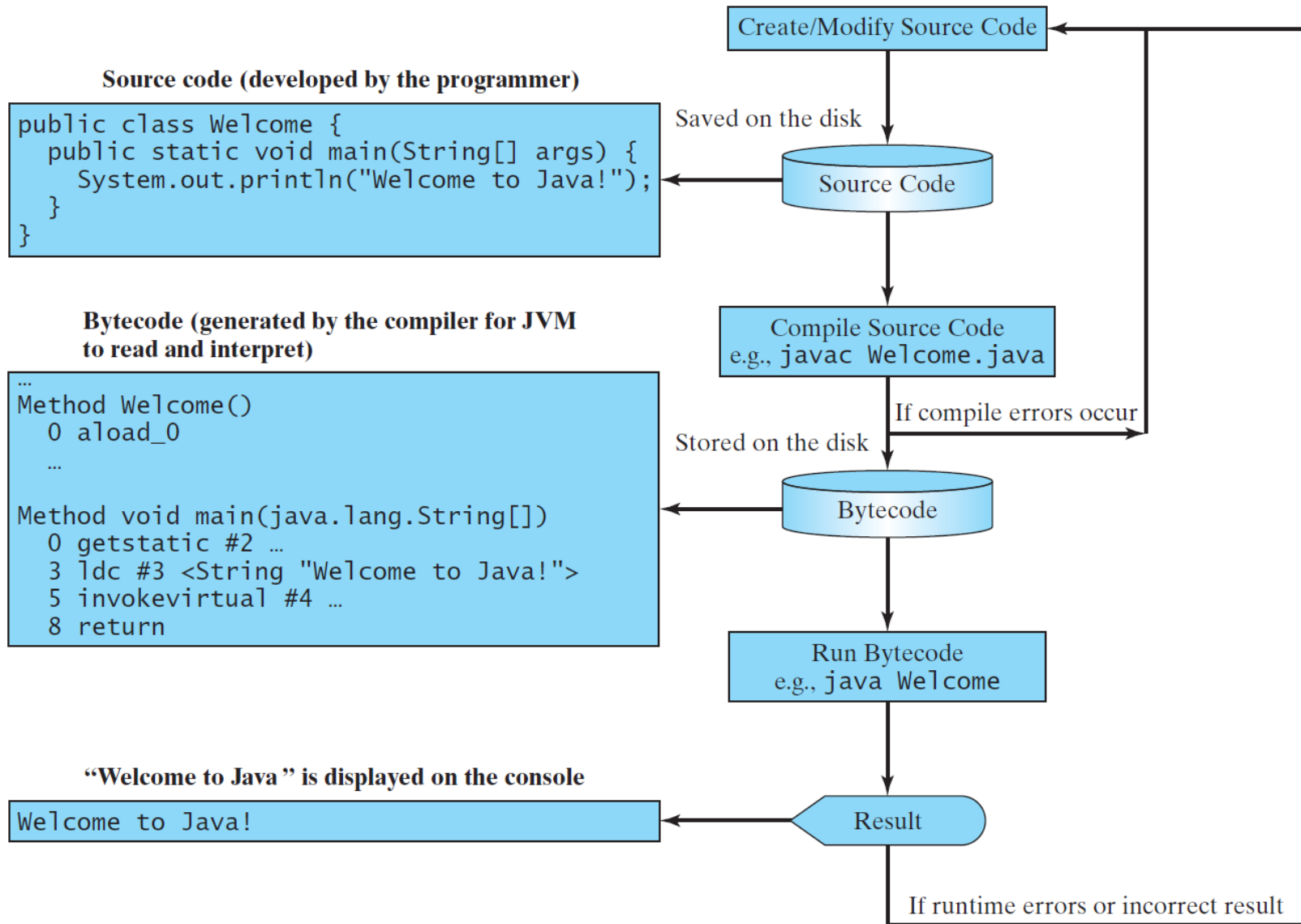


Lecture 2

Program execution, variables, control flow, io



- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks



- Every Java program must have at least **one class**.
- Each class has a name.
- By convention, class names start with an uppercase letter.
- The class name **must** coincide with the file name (case sensitive).

```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```



- Line 2 defines the main method.
- In order to run a class, the class must contain a method named main.
- The program is executed from the main method.

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

- A statement represents an action or a sequence of actions.
- The statement `System.out.println("Welcome to Java!")` displays "Welcome to Java!".

```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

- Every statement in Java ends with a semicolon ;

```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

- Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program.
- For example, when the compiler sees the word `class`, it understands that the word after `class` is the name for the class.

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```




- A pair of braces in a program forms a block that groups components of a program.
- Equivalent to tabs in python

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



- A pair of braces in a program forms a block that groups components of a program.
- Equivalent to tabs in python

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



Character	Name	Description
{ }	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.



```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```



```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```




- **Appropriate Comments**
 - Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
 - Include your name, class section, instructor, date, and a brief description at the beginning of the program.
- **Naming Conventions**
 - Choose meaningful and descriptive names
 - Use CamelCase for class names
- **Proper Indentation and Spacing Lines**
 - Indent the code like python
 - Use blank line to separate segments of the code
- **Block Styles**
 - Next line
 - End of line



```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```



Enter main method

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



Execute statement

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



Print message

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

- Syntax Errors
 - Detected by the compiler
- Runtime Errors
 - Causes the program to abort
- Logic Errors
 - Produces incorrect result

```
// This program contains errors
public class ShowSyntaxErrors {
    public static main(String[] args) {
        System.out.println("Welcome to Java");
    }
}
```

```
// This program contains errors
public class ShowRuntimeErrors {
    public static void main(String[] args) {
        System.out.println(1 / 0);
    }
}
```




```
// This program contains errors
public class ShowLogicErrors {
    public static void main(String[] args) {
        System.out.println("Celsius 35 is
        Fahrenheit degree ");
        System.out.println((9 / 5) * 35 + 32);
    }
}
```

■ Declaring

- `int x;` // Declare x to be an integer variable;
- `double radius;` // Declare radius to be a double variable;
- `char a;` // Declare a to be a character variable;

■ Assign

- `x = 1;` // Assign 1 to x;
- `radius = 1.0;` // Assign 1.0 to radius;
- `a = 'A';` // Assign 'A' to a;

■ Constants

- `final double PI = 3.14159;`
- `final int SIZE = 3;`

■ Declaring and initializing in one step

- `int x = 1;`
- `double d = 1.4;`

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs (\$).
- An identifier must start with a letter, an underscore (_), or a dollar sign (\$). It cannot start with a digit.
- An identifier cannot be a reserved word.
- An identifier cannot be true, false, or null.
- An identifier can be of any length.

- Naming conventions
 - Choose meaningful and descriptive names.
 - Use lowercase. If the name consists of several words, use camelCase
 - Constants, capitalize all letters, and use underscores to connect words.

Name	Range	Storage Size
byte	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
short	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
int	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
long	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Name	Meaning	Example	Result
+	Addition	$34 + 1$	35
-	Subtraction	$34.0 - 0.1$	33.9
*	Multiplication	$300 * 30$	9000
/	Division	$1.0 / 2.0$	0.5
%	Remainder	$20 \% 3$	2

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Equivalent</i>
<code>+=</code>	Addition assignment	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Subtraction assignment	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Multiplication assignment	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Division assignment	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	Remainder assignment	<code>i %= 8</code>	<code>i = i % 8</code>



<i>Operator</i>	<i>Name</i>	<i>Description</i>	<i>Example (assume i = 1)</i>
++var	preincrement	Increment var by 1 , and use the new var value in the statement	int j = ++i; // j is 2, i is 2
var++	postincrement	Increment var by 1 , but use the original var value in the statement	int j = i++; // j is 1, i is 2
--var	predecrement	Decrement var by 1 , and use the new var value in the statement	int j = --i; // j is 0, i is 0
var--	postdecrement	Decrement var by 1 , and use the original var value in the statement	int j = i--; // j is 1, i is 0



```
int i = 10;
```

```
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

```
int i = 10;
```

```
int newNum = 10 * (++i);
```

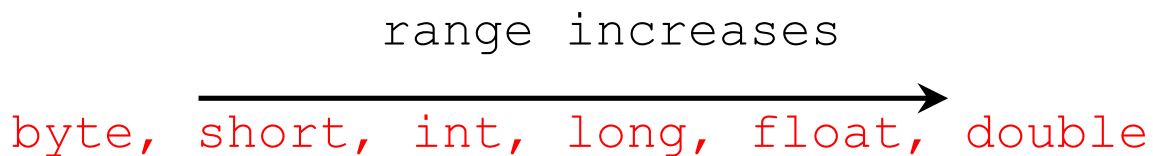
Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read.
- Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this
- ```
int k = ++i + i
```



- When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:
  - If one of the operands is double, the other is converted into double.
  - Otherwise, if one of the operands is float, the other is converted into float.
  - Otherwise, if one of the operands is long, the other is converted into long.
  - Otherwise, both operands are converted into int.
  
- Implicit casting
  - `double d = 3;` (type widening)
- Explicit casting
  - `int i = (int)3.0;` (type narrowing)
  - `int i = (int)3.9;` (Fraction part is truncated)



- **Undeclared/Uninitialized Variables and Unused Variables**

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

- **Integer Overflow**

```
int value = 2147483647 + 1; // value will be -2147483648
```

- **Round-off Errors**

```
System.out.println(1.0 - 0.9); //prints 0.099999999999999998
```

- **Unintended Integer Division**

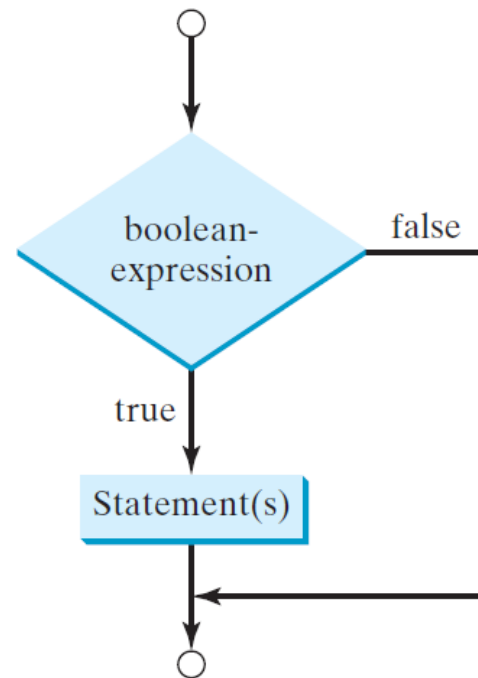
```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
```

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
```

- Often in a program you need to compare two values, such as whether *i* is greater than *j*. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.
- `boolean b = (1 > 2);`

| Java Operator      | Mathematics Symbol | Name                     | Example (radius is 5)       | Result             |
|--------------------|--------------------|--------------------------|-----------------------------|--------------------|
| <code>&lt;</code>  | <code>&lt;</code>  | less than                | <code>radius &lt; 0</code>  | <code>false</code> |
| <code>&lt;=</code> | <code>≤</code>     | less than or equal to    | <code>radius &lt;= 0</code> | <code>false</code> |
| <code>&gt;</code>  | <code>&gt;</code>  | greater than             | <code>radius &gt; 0</code>  | <code>true</code>  |
| <code>&gt;=</code> | <code>≥</code>     | greater than or equal to | <code>radius &gt;= 0</code> | <code>true</code>  |
| <code>==</code>    | <code>=</code>     | equal to                 | <code>radius == 0</code>    | <code>false</code> |
| <code>!=</code>    | <code>≠</code>     | not equal to             | <code>radius != 0</code>    | <code>true</code>  |

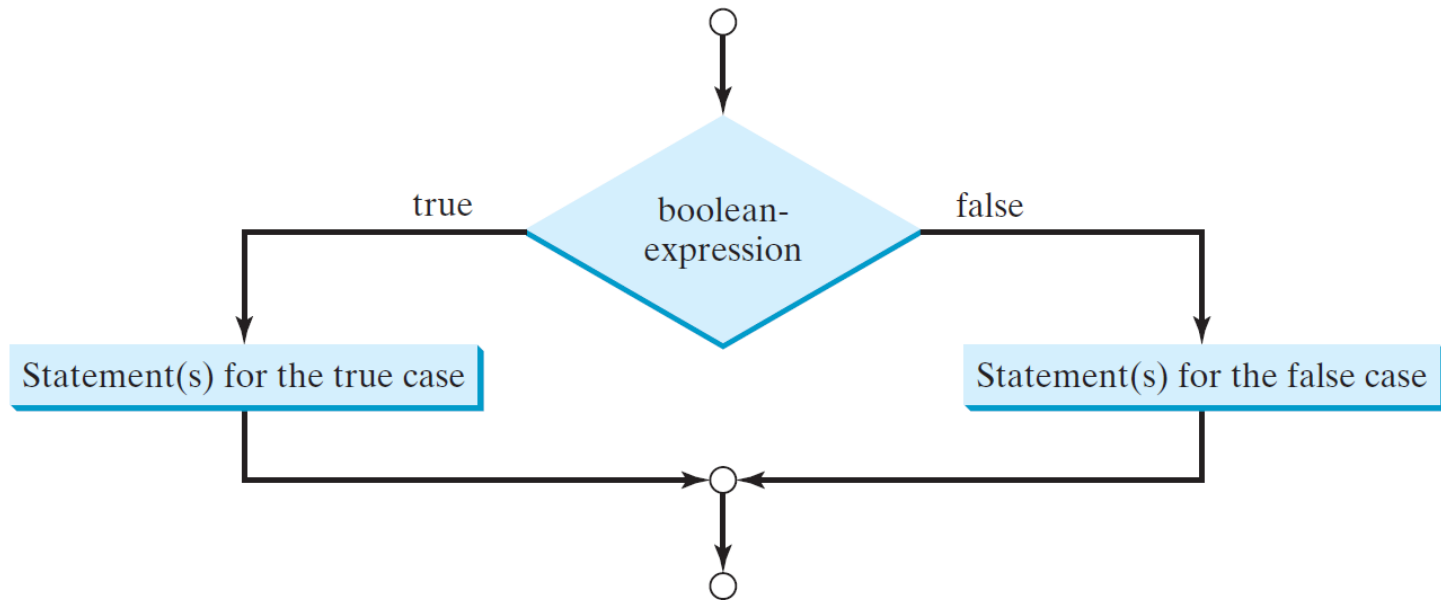
```
if (boolean-expression) {
 statement(s);
}
```



```

if (boolean-expression) {
 statement(s)-for-the-true-case;
}
else {
 statement(s)-for-the-false-case;
}

```



```
if (radius >= 0) {
 area = radius * radius * 3.14159;
 System.out.println("The area is " + area);
}
else {
 System.out.println("Negative input");
}
```

```
if (score >= 90.0)
 System.out.print("A");
else
 if (score >= 80.0)
 System.out.print("B");
 else
 if (score >= 70.0)
 System.out.print("C");
 else
 if (score >= 60.0)
 System.out.print("D");
 else
 System.out.print("F");
```

==

```
if (score >= 90.0)
 System.out.print("A");
else if (score >= 80.0)
 System.out.print("B");
else if (score >= 70.0)
 System.out.print("C");
else if (score >= 60.0)
 System.out.print("D");
else
 System.out.print("F");
```



```
switch (status) {
 case 0: statemet-for-0;
 break;
 case 1: statemet-for-1;
 break;
 case 2: statemet-for-2;
 break;
 case 3: statemet-for-3;
 break;
 default: statemet-for-1-3;
}
```

1. `var++`, `var--`
2. `+`, `-` (Unary plus and minus), `++var`, `--var`
3. `(type)` Casting
4. `!` (Not)
5. `*`, `/`, `%` (Multiplication, division, and remainder)
6. `+`, `-` (Binary addition and subtraction)
7. `<`, `<=`, `>`, `>=` (Relational operators)
8. `==`, `!=`; (Equality)
9. `^` (Exclusive OR)
10. `&&` (Conditional AND) Short-circuit AND
11. `||` (Conditional OR) Short-circuit OR
12. `=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)





- Create a Scanner object

```
Scanner input = new Scanner(System.in);
```

- Use a method to obtain to a value.

| Method                    | Description                                |
|---------------------------|--------------------------------------------|
| <code>nextByte()</code>   | reads an integer of the <b>byte</b> type.  |
| <code>nextShort()</code>  | reads an integer of the <b>short</b> type. |
| <code>nextInt()</code>    | reads an integer of the <b>int</b> type.   |
| <code>nextLong()</code>   | reads an integer of the <b>long</b> type.  |
| <code>nextFloat()</code>  | reads a number of the <b>float</b> type.   |
| <code>nextDouble()</code> | reads a number of the <b>double</b> type.  |

- Example

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```



```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
```

```
Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```