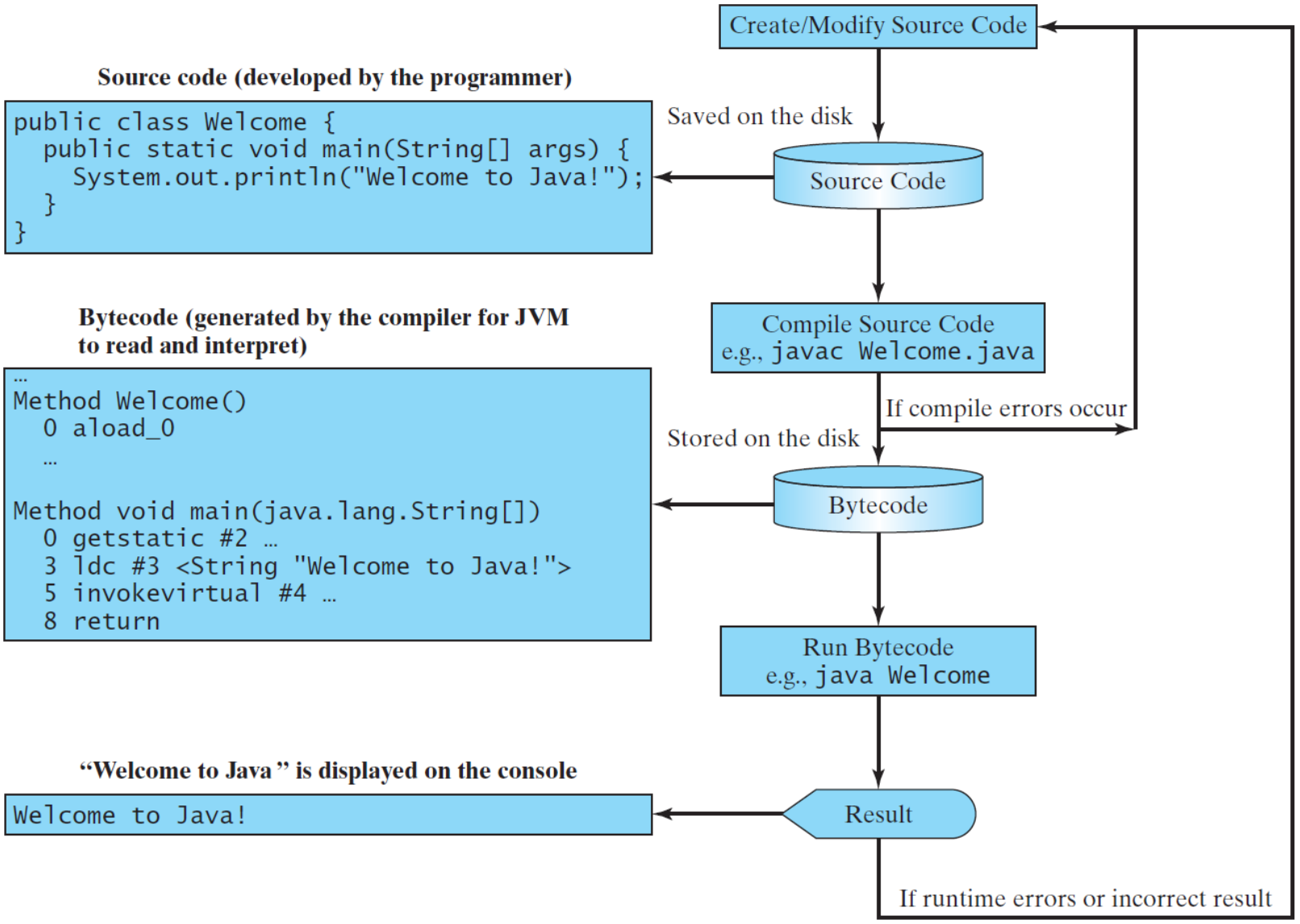
# Lecture 2

Program execution, variables, control flow, io

# Creating, Compiling, and Running Programs



# Anatomy of a Java Program

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

# Class Name

- 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!");
  }
}
```

# Main Method

- 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!");
  }
}
```

# Statement

- 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!");
  }
}
```

# Statement Terminator

- 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

- 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!");
  }
}
```

# Blocks

- 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!");
  }
}
```

# Blocks

- 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!");
  }
}
```

# Special Symbols

| 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!");
  }
}
```

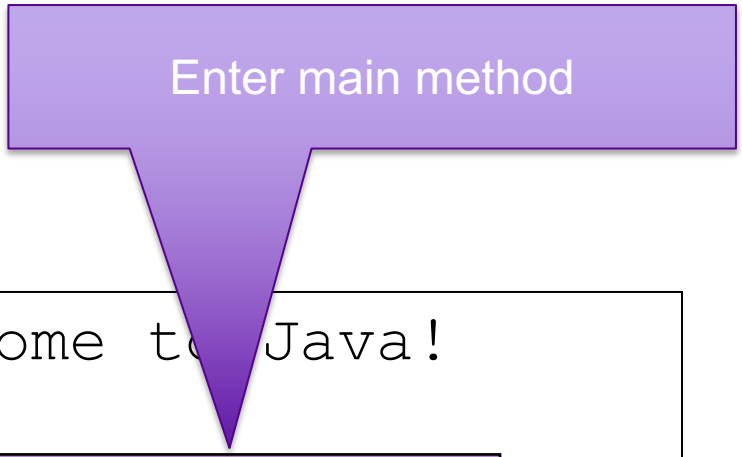
# Programming Style and Documentation

- 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

# Trace a Program Execution

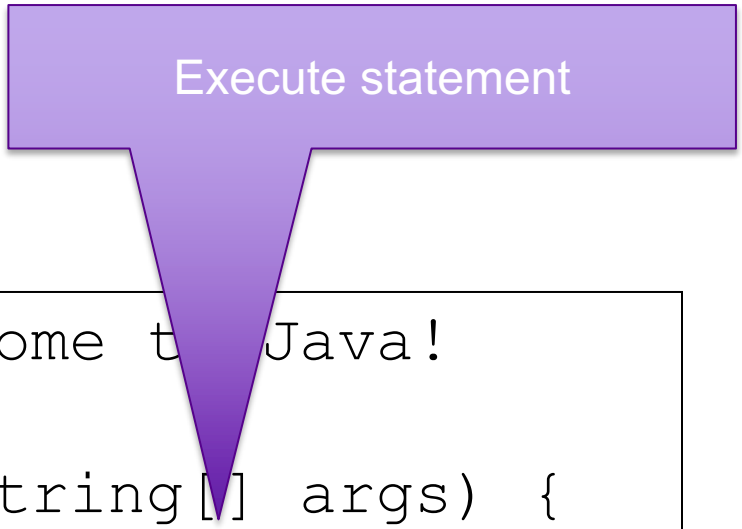
```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

# Trace a Program Execution



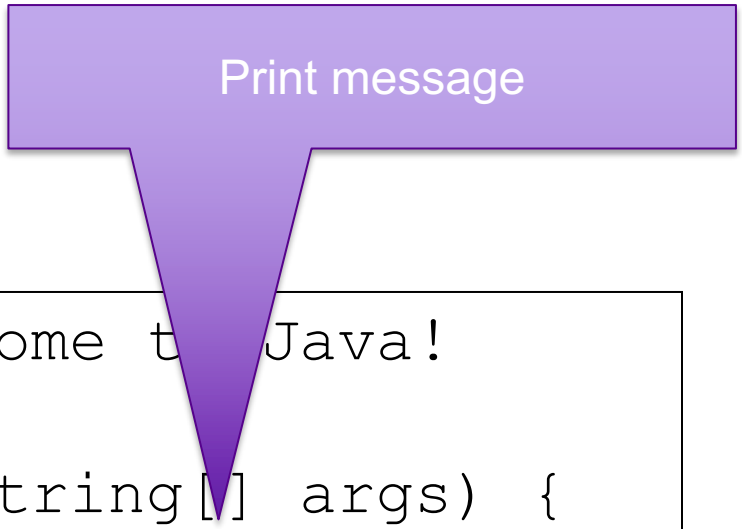

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

# Trace a Program Execution




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

# Trace a Program Execution




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

# Programming Errors

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

# Syntax Errors

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

# Runtime Errors

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

# Logic Errors

```
// 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);
  }
}
```

# Variables

- 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;`

# Variable names AKA Identifiers

- 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.

# Types

| 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 $2^{63} - 1$<br>(i.e., -9223372036854775808 to 9223372036854775807) | 64-bit signed |
| `float` | Negative range:<br>-3.4028235E+38 to -1.4E-45<br>Positive range:<br>1.4E-45 to 3.4028235E+38 | 32-bit IEEE 754 |
| `double` | Negative range:<br>-1.7976931348623157E+308 to -4.9E-324<br><br>Positive range:<br>4.9E-324 to 1.7976931348623157E+308 | 64-bit IEEE 754 |

# Numeric Operators

| 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 |

# Augmented Assignment Operators

| *Operator* | *Name* | *Example* | *Equivalent* |
|---|---|---|---|
| `+=` | Addition assignment | `i += 8` | `i = i + 8` |
| `-=` | Subtraction assignment | `i -= 8` | `i = i - 8` |
| `*=` | Multiplication assignment | `i *= 8` | `i = i * 8` |
| `/=` | Division assignment | `i /= 8` | `i = i / 8` |
| `%=` | Remainder assignment | `i %= 8` | `i = i % 8` |

# Increment and Decrement Operators

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

# Increment and Decrement Operators, cont.

```
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

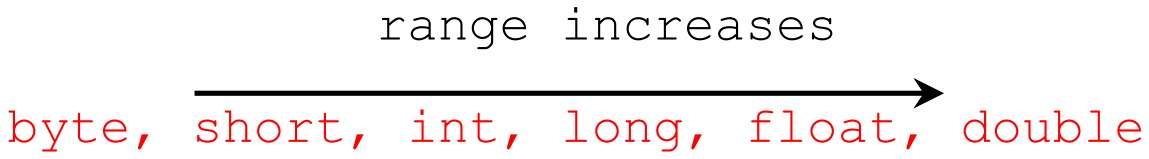
```
i = i + 1;
int newNum = 10 * i;
```

- 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`

# Conversion Rules

- 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)



# Common Errors and Pitfalls

- 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 0.09999999999999998
```

- 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;
```

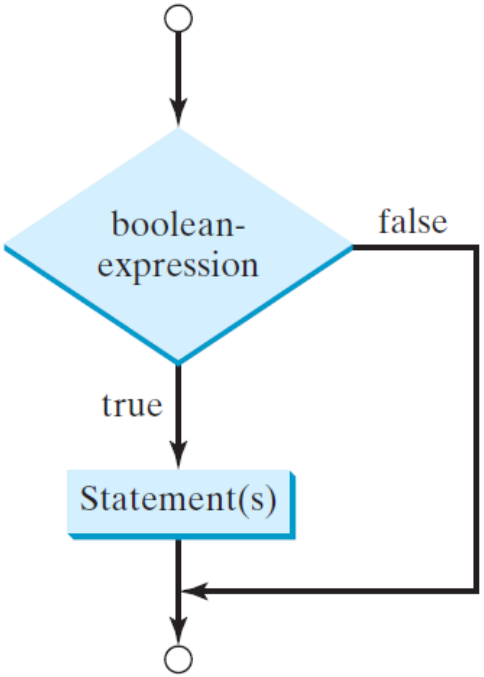
# The boolean Type and Operators

- 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 |
|---|---|---|---|---|
| `<` | < | less than | `radius < 0` | `false` |
| `<=` | ≤ | less than or equal to | `radius <= 0` | `false` |
| `>` | > | greater than | `radius > 0` | `true` |
| `>=` | ≥ | greater than or equal to | `radius >= 0` | `true` |
| `==` | = | equal to | `radius == 0` | `false` |
| `!=` | ≠ | not equal to | `radius != 0` | `true` |

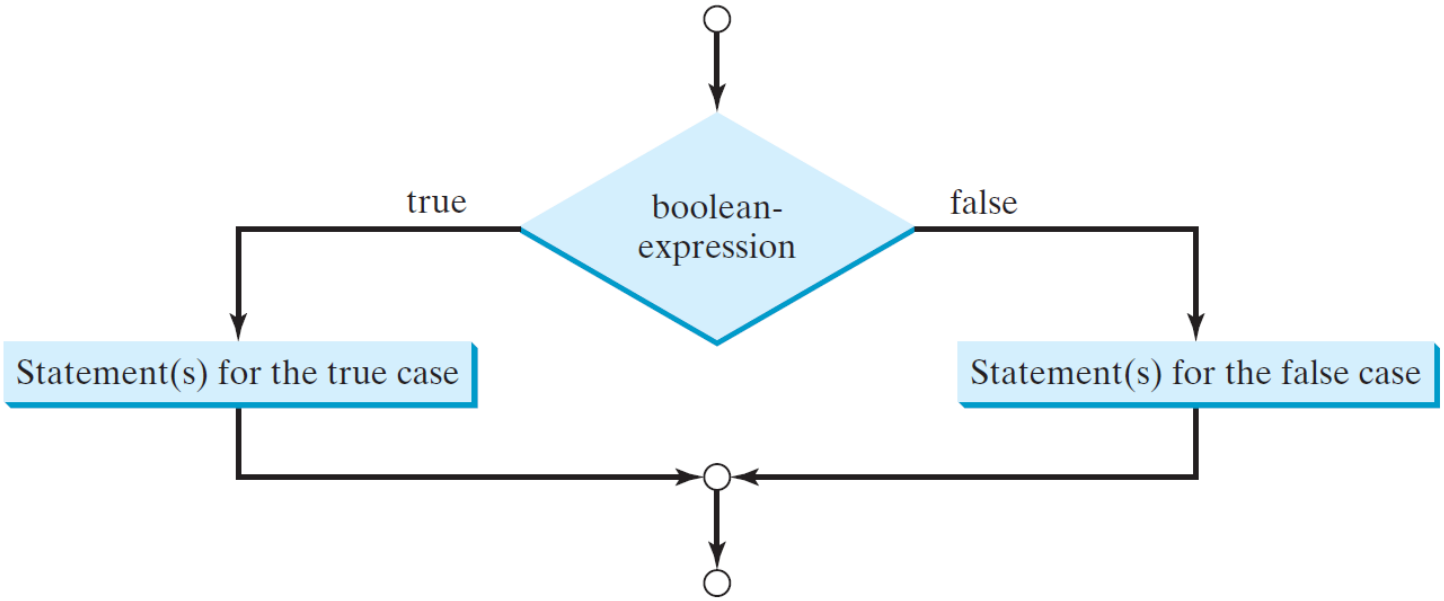
# One-way if Statements

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



# Two-way if Statements

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



# if-else Example

```
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 Statements

```
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;
}
```

# Operator Precedence

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)

# Reading Input from the Console

- Create a Scanner object

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

- Use a method to obtain to a value.

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

- Example

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

# Common Pitfall: Redundant Input Objects

```
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();
```