# Variable $\approx$ Box

# Variable Declaration

- You give a name for the variable, say x.
- Additionally, you need to assign a type for the variable.
- For example,

```
...
        int x; // x is a variable declared an interger type.
...
```

- Variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.[1]
- It is worth to mention that, the date type determines the size, which is measured in bytes[2].

[1] Actually, all declared variables are created at the compile time.

[2] 1 byte = 8 bits; bit = binary digit.

# Naming Rules

- Identifiers are the names that identify the elements such as variables, methods, and classes in the program.
- The naming rule excludes the following situations:
  - cannot start with a digit
  - cannot be any reserved word[3]
  - cannot include any blank between letters
  - cannot contain $+, -, *, /$ and %
- Note that Java is case sensitive[4].

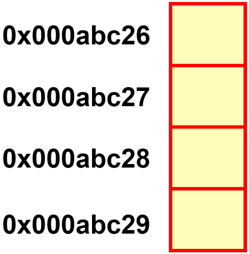
[3] See the next page.
[4] The letter A and a are different.

# Reserved Words[5]

| | | | |
|---|---|---|---|
| `abstract` | `double` | `int` | `super` |
| `assert` | `else` | `interface` | `switch` |
| `boolean` | `enum` | `long` | `synchronized` |
| `break` | `extends` | `native` | `this` |
| `byte` | `final` | `new` | `throw` |
| `case` | `finally` | `package` | `throws` |
| `catch` | `float` | `private` | `transient` |
| `char` | `for` | `protected` | `try` |
| `class` | `goto` | `public` | `void` |
| `const` | `if` | `return` | `volatile` |
| `continue` | `implements` | `short` | `while` |
| `default` | `import` | `static` | |
| `do` | `instanceof` | `strictfp`* | |

[5] See Appendix A in YDL, p. 1253.

# Variable as Alias of Memory Address




- The number 0x000abc26 stands for one memory address in hexadecimal (0-9, and a-f).[6]
- The variable $x$ itself refers to 0x000abc26 in the program after compilation.

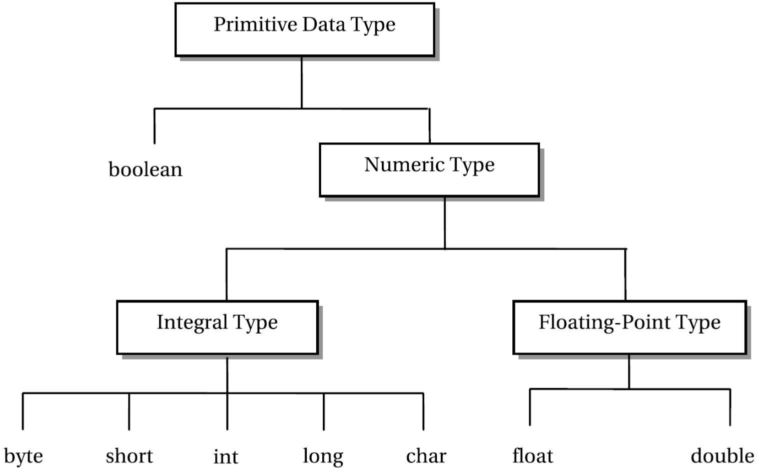
[6] See `https://en.wikipedia.org/wiki/Hexadecimal`.

# Data Types

- Java is a strongly typed[7] programming language.
- Every variable has a type.
- Also, every (mathematical) expression has a type.
- There are two categories of data types: primitive data types, and reference data types.

[7] You cannot change the type of the variable after declaration.

# Primitive Data Types[8]




[8] See Figure 3-4 in Sharan, p. 67.

# Integers

| Name | Width | Range |
|---|---|---|
| **long** | 64 | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| **int** | 32 | –2,147,483,648 to 2,147,483,647 |
| **short** | 16 | –32,768 to 32,767 |
| **byte** | 8 | –128 to 127 |

- The most commonly used integer type is int.
- If the integer values are larger than its feasible range, then an overflow occurs.

# Floats

| Name | Width in Bits | Approximate Range |
| --- | --- | --- |
| **double** | 64 | 4.9e–324 to 1.8e+308 |
| **float** | 32 | 1.4e–045 to 3.4e+038 |

- Floats are used when evaluating expressions that require fractional precision.
  - For example, sin(), cos(), and sqrt().
- The performance for the double values is actually faster than that for float values on modern processors that have been optimized for high-speed mathematical calculations.
- Be aware that floating-point arithmetic can only approximate real arithmetic.[9] (Why?)

[9] See `https://en.wikipedia.org/wiki/Numerical_error`.

# Example: $0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0$?

```java
public class FloatsDemo {
    public static void main(String[] args) {
        System.out.println(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
    }
}
```

- The result is surprising. (Why?)
- You may try this decimal-binary converter.
- This issue occurs not only in decimal numbers, but also big integers represented in floats.[10]
- So the floats are not reliable unless the algorithm is designed elaborately for numerical errors.[11]

[10] Thanks to a lively discussion on June 26, 2016.

[11] See https://finance.technews.tw/2017/01/10/largan-stock-trouble/.

# Example: Loss of Significance

- For example,

```
...
        System.out.println(3.14 + 1e20 − 1e20); // output ?
        System.out.println(3.14 + (1e20 − 1e20)); // output ?
...
```

- Can you explain why?

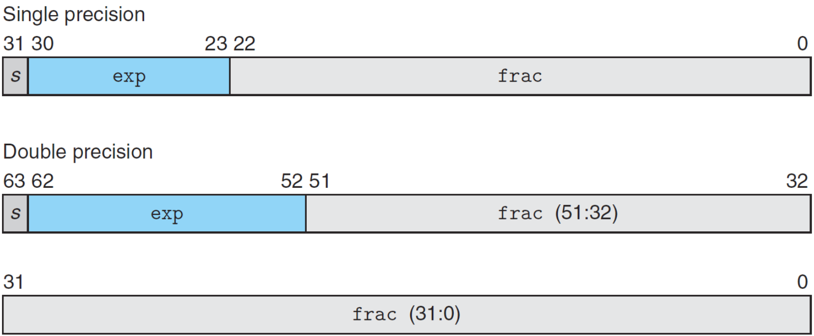
# IEEE Floating-Point Representation[12]

$$x = (-1)^s \times M \times 2^E$$

- The sign $s$ determines whether the number is negative ($s = 1$) or positive ($s = 0$).
- The significand $M$ is a fractional binary number that ranges either between 1 and $2 - \epsilon$, or between 0 and $1 - \epsilon$.
- The exponent $E$ weights the value by a (possibly negative) power of 2.

[12] William Kahan (1985); Aka IEEE754.

# Illustration[13]




- That is why we call a double value.

[13] See Figure 2-31 in Byrant, p. 104.

# Assignments

- An assignment statement designates a value to the variable.

```
int x; // make a variable declaration
...
x = 1; // assign 1 to x
```

- The equal sign ($=$) is used as the assignment operator.
  - For example, is the expression $x = x + 1$ correct?
  - Direction: from the right-hand side to the left-hand side
- To assign a value to a variable, you must place the variable name to the left of the assignment operator.[14]
  - For example, $1 = x$ is wrong.
  - 1 cannot be resolved to a memory space.

[14] x can be a l-value and r-value, but 1 and other numbers can be only r-value but not l-value. See Value.

# Two "Before" Rules

- Every variable has a scope.
  - The scope of a variable is the range of the program where the variable can be referenced.[15]
- A variable must be declared before it can be assigned a value.
  - In practice, do not declare the variable until you need it.
- A declared variable must be assigned a value before it can be used.[16]

[15] The detail of variable scope is introduced later.

[16] In symbolic programming, such as Mathematica and Maple, a variable can be manipulated without assigning a value. For example, $x + x$ returns $2x$.

# Arithmetic Operators[17]

| *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 |

- Note that the operator depends on the operands involved.

[17] See Table 2-3 in YDL, p. 46.

# Tricky Pitfalls

- Can you explain this result?

```java
...
        double x = 1 / 2;
        System.out.println(x); // output?
...
```

- Revisit $0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0$.[18]

```java
...
        System.out.println(1 / 2 - 1 / 10 - 1 / 10 - 1 / 10 - 1
            / 10 - 1 / 10); // output 0; however, this is not
            the real solution to the original problem.
...
```

[18] Thanks to a lively discussion on on June 7, 2016.

# Type Conversion and Compatibility

- If a type is compatible to another, then the compiler will perform the conversion implicitly.
  - For example, the integer 1 is compatible to a double value 1.0.
- However, there is no automatic conversion from double to int. (Why?)
- To do so, you must use a **cast**, which performs an explicit conversion for compilation.
- Similarly, a long value is not compatible to int.

# Casting

```java
...
        int x = 1;
        double y = x; // compatible; implicit conversion
        x = y; // incompatible; need an explicit conversion by
            casting
        x = (int) y; // succeed!!
...
```

- Note that the Java compiler does only type-checking but no real execution before compilation.
- In other words, the values of *x* and *y* are unknown until they are really executed.

# Type Conversion and Compatibility (concluded)

- small-size types $\rightarrow$ large-size types
- small-size types $\not\leftarrow$ large-size types (need a cast)
- simple types $\rightarrow$ complicated types
- simple types $\not\leftarrow$ complicated types (need a cast)

# Characters

- A character stored by the machine is represented by a sequence of 0's and 1's.
  - For example, ASCII code. (See the next page.)
- The char type is a 16-bit unsigned primitive data type.[19]

[19] Java uses Unicode to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.

# ASCII (7-bit version)

| Hex | Dec | Char | | Hex | Dec | Char | Hex | Dec | Char | Hex | Dec | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00 | 0 | NULL | null | 0x20 | 32 | Space | 0x40 | 64 | @ | 0x60 | 96 | ` |
| 0x01 | 1 | SOH | Start of heading | 0x21 | 33 | ! | 0x41 | 65 | A | 0x61 | 97 | a |
| 0x02 | 2 | STX | Start of text | 0x22 | 34 | " | 0x42 | 66 | B | 0x62 | 98 | b |
| 0x03 | 3 | ETX | End of text | 0x23 | 35 | # | 0x43 | 67 | C | 0x63 | 99 | c |
| 0x04 | 4 | EOT | End of transmission | 0x24 | 36 | $ | 0x44 | 68 | D | 0x64 | 100 | d |
| 0x05 | 5 | ENQ | Enquiry | 0x25 | 37 | % | 0x45 | 69 | E | 0x65 | 101 | e |
| 0x06 | 6 | ACK | Acknowledge | 0x26 | 38 | & | 0x46 | 70 | F | 0x66 | 102 | f |
| 0x07 | 7 | BELL | Bell | 0x27 | 39 | ' | 0x47 | 71 | G | 0x67 | 103 | g |
| 0x08 | 8 | BS | Backspace | 0x28 | 40 | ( | 0x48 | 72 | H | 0x68 | 104 | h |
| 0x09 | 9 | TAB | Horizontal tab | 0x29 | 41 | ) | 0x49 | 73 | I | 0x69 | 105 | i |
| 0x0A | 10 | LF | New line | 0x2A | 42 | * | 0x4A | 74 | J | 0x6A | 106 | j |
| 0x0B | 11 | VT | Vertical tab | 0x2B | 43 | + | 0x4B | 75 | K | 0x6B | 107 | k |
| 0x0C | 12 | FF | Form Feed | 0x2C | 44 | , | 0x4C | 76 | L | 0x6C | 108 | l |
| 0x0D | 13 | CR | Carriage return | 0x2D | 45 | - | 0x4D | 77 | M | 0x6D | 109 | m |
| 0x0E | 14 | SO | Shift out | 0x2E | 46 | . | 0x4E | 78 | N | 0x6E | 110 | n |
| 0x0F | 15 | SI | Shift in | 0x2F | 47 | / | 0x4F | 79 | O | 0x6F | 111 | o |
| 0x10 | 16 | DLE | Data link escape | 0x30 | 48 | 0 | 0x50 | 80 | P | 0x70 | 112 | p |
| 0x11 | 17 | DC1 | Device control 1 | 0x31 | 49 | 1 | 0x51 | 81 | Q | 0x71 | 113 | q |
| 0x12 | 18 | DC2 | Device control 2 | 0x32 | 50 | 2 | 0x52 | 82 | R | 0x72 | 114 | r |
| 0x13 | 19 | DC3 | Device control 3 | 0x33 | 51 | 3 | 0x53 | 83 | S | 0x73 | 115 | s |
| 0x14 | 20 | DC4 | Device control 4 | 0x34 | 52 | 4 | 0x54 | 84 | T | 0x74 | 116 | t |
| 0x15 | 21 | NAK | Negative ack | 0x35 | 53 | 5 | 0x55 | 85 | U | 0x75 | 117 | u |
| 0x16 | 22 | SYN | Synchronous idle | 0x36 | 54 | 6 | 0x56 | 86 | V | 0x76 | 118 | v |
| 0x17 | 23 | ETB | End transmission block | 0x37 | 55 | 7 | 0x57 | 87 | W | 0x77 | 119 | w |
| 0x18 | 24 | CAN | Cancel | 0x38 | 56 | 8 | 0x58 | 88 | X | 0x78 | 120 | x |
| 0x19 | 25 | EM | End of medium | 0x39 | 57 | 9 | 0x59 | 89 | Y | 0x79 | 121 | y |
| 0x1A | 26 | SUB | Substitute | 0x3A | 58 | : | 0x5A | 90 | Z | 0x7A | 122 | z |
| 0x1B | 27 | FSC | Escape | 0x3B | 59 | ; | 0x5B | 91 | [ | 0x7B | 123 | { |
| 0x1C | 28 | FS | File separator | 0x3C | 60 | < | 0x5C | 92 | \ | 0x7C | 124 | \| |
| 0x1D | 29 | GS | Group separator | 0x3D | 61 | = | 0x5D | 93 | ] | 0x7D | 125 | } |
| 0x1E | 30 | RS | Record separator | 0x3E | 62 | > | 0x5E | 94 | ^ | 0x7E | 126 | ~ |
| 0x1F | 31 | US | Unit separator | 0x3F | 63 | ? | 0x5F | 95 | _ | 0x7F | 127 | DEL |

# Example

- Characters can also be used as (positive) integers on which you can perform arithmetic operations.[20]
- For example,

```
...
        char x = 'a';  // single-quoted: a char value
        System.out.println(x + 1); // output 98!!
        System.out.println((char)(x + 1)); // output b

        String s = "Java"; // double-quoted: a String object
...
```

- You can imagine that a String object comprises characters equipped with plentiful tools.[21]

[20] See https://en.wikipedia.org/wiki/Cryptography.

[21] As an analogy, a molecule (string) consists of atoms (characters).

# Boolean Values

- The program is supposed to do decision making by itself, for example, Google Driverless Car.[22]
- To do this, Java has the boolean-type flow controls (selections and iterations).
- This type has only two possible values, true and false.
- Note that a boolean value cannot be cast into a value of another type, nor can a value of another type be cast into a boolean value. (Why?)

[22] See https://www.google.com/selfdrivingcar/.

# Rational Operators[23]

| *Java Operator* | *Mathematics Symbol* | *Name* |
|---|---|---|
| < | $<$ | less than |
| <= | $\leq$ | less than or equal to |
| > | $>$ | greater than |
| >= | $\geq$ | greater than or equal to |
| == | $=$ | equal to |
| != | $\neq$ | not equal to |

- These operators take two operands.
- Rational expressions return a boolean value.
- Note that the equality operator is double equality sign (==), not single equality sign (=).

[23] See Table 3-1 in YDL, p. 82.

# Example

```java
...
        int x = 2;
        boolean a = x > 1;
        boolean b = x < 1;
        boolean c = x == 1;
        boolean d = x != 1;
        boolean e = 1 < x < 3; // sorry?
...
```

- Be aware that e is logically correct but syntactically wrong.
- Usually, the boolean expression consists of a combination of rational expressions.
    - For example, $1 < x < 3$ should be $(1 < x)\&\&(x < 3)$, where && refers to the AND operator.

# Logical Operators[24]

| *Operator* | *Name* | *Description* |
|---|---|---|
| ! | not | logical negation |
| && | and | logical conjunction |
| \|\| | or | logical disjunction |
| ^ | exclusive or | logical exclusion |

[24]See Table 3-2 in YDL, p. 102.

# Truth Table

- Let $X$ and $Y$ be two Boolean variables.
- Then the truth table for logical operators is as follows:

| X | Y | !X | X&&Y | X \|\| Y | X $\wedge Y$ |
|---|---|---|---|---|---|
| T | T | F | T | T | F |
| T | F | F | F | T | T |
| F | T | T | F | T | T |
| F | F | T | F | F | F |

- Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.[25]

[25] See any textbook for digital circuit design.

*"Logic is the anatomy of thought."*

– John Locke (1632–1704)

*"This sentence is false."*

– anonymous

*"I know that I know nothing."*

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)

# Arithmetic Compound Assignment Operators

| | |
|---|---|
| ++ | Increment |
| += | Addition assignment |
| − = | Subtraction assignment |
| *= | Multiplication assignment |
| /= | Division assignment |
| %= | Modulus assignment |
| − − | Decrement |

- Note that these shorthand operators are not available in languages such as Matlab, R, and Python.

# Example

```java
...
        int x = 1;
        System.out.println(x); // output 1
        x = x + 1;
        System.out.println(x); // output 2
        x += 2;
        System.out.println(x); // output 4
        x++; // equivalent to x += 1 and x = x + 1
        System.out.println(x); // output 5
...
```

- The compound assignment operators are also useful for char values.[26]
- For example,

```java
...
        char s = 'a';
        System.out.println(s); // output a
        s += 1;
        System.out.println(s); // output b
        s++;
        System.out.println(s); // output c
...
```

[26] Contribution by Mr. Edward Wang (Java265) on May 1, 2016.

# ++x vs. x++

- The expression ++x first increments the value of x and then returns x.
- Instead, the expression x++ first returns the value of x and then increments itself.
- For example,

```java
...
        int x = 1;
        int y = ++x;
        System.out.println(y); // output 2; aka preincrement
        System.out.println(x); // output 2

        int w = 1;
        int z = w++;
        System.out.println(z); // output 1; aka postincrement
        System.out.println(w); // output 2
...
```

- We will use these notations very often.

# Operator Precedence[27]

| *Precedence* | *Operator* |
|---|---|
| | `var++` and `var--` (Postfix) |
| | `+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix) |
| | (type) (Casting) |
| | `!` (Not) |
| | `*`, `/`, `%` (Multiplication, division, and remainder) |
| | `+`, `-` (Binary addition and subtraction) |
| | `<`, `<=`, `>`, `>=` (Comparison) |
| | `==`, `!=` (Equality) |
| | `^` (Exclusive OR) |
| | `&&` (AND) |
| | `\|\|` (OR) |
| | `=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator) |

[27] See Table3-10 in YDL, p. 116.

# Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.

# Scanner Objects

- It is not convenient to modify the source code and recompile it for a different radius.
- Reading from the console enables the program to receive an input from the user.
- A **Scanner** object provides some input methods, say the input received from the keyboard or the files.
- Java uses **System.in** to refer to the standard input device, by default, the keyboard.

# Example: Reading Input From The Console

Write a program which receives a number as input, and outputs the area of the circle.

```
import java.util.Scanner;
...
        Scanner input = new Scanner(System.in);
        System.out.println("Enter r?");
        // input
        int r = input.nextInt();
        // algorithm
        double area = r * r * 3.14;
        // output
        System.out.println(area);
        input.close();
...
```

- In the listing, Line 3 is to create a **Scanner** object by the new operator, as an agent between the keyboard and your program.
- Note that all objects are resided in the heap of the memory.
- To control this object, its memory address is then assigned to the variable *input* which is a variable in the stack of memory.
- So the variable *input* is a reference.
- We will discuss the objects and reference variables later.

# Methods Provided by Scanner Objects[28]

| *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. |
| `next()` | reads a string that ends before a whitespace character. |
| `nextLine()` | reads a line of text (i.e., a string ending with the *Enter* key pressed). |

[28] See Table 2-1 in YDL, p. 38.

# Example: Mean and Standard Deviation

Write a program which calculates the mean and the standard deviation of 3 numbers.

- The mean of 3 numbers is given by $\overline{x} = \left(\sum_{i=1}^{3} x_i\right)/3$.
- Also, the resulting standard deviation is given by

$$S = \sqrt{\frac{\sum_{i=1}^{3}(x_i - \overline{x})^2}{3}}.$$

- You may use these two methods:
  - Math.pow(double x , double y) for $x^y$
  - Math.sqrt(double x) for $\sqrt{x}$
- See more methods within Math class.

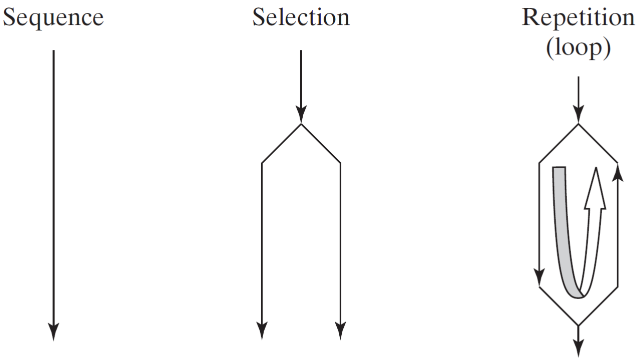
```java
...
        Scanner input = new Scanner(System.in);
        System.out.println("a = ?");
        double a = input.nextDouble();
        System.out.println("b = ?");
        double b = input.nextDouble();
        System.out.println("c = ?");
        double c = input.nextDouble();

        double mean = (a + b + c) / 3;
        double std = Math.sqrt((Math.pow(a - mean, 2) +
                                Math.pow(b - mean, 2) +
                                Math.pow(c - mean, 2)) / 3);

        System.out.println("mean = " + mean);
        System.out.println("std = " + std);
...
```

```java
class Lecture3 {

                    "Selections"

}

// Keywords
if, else, else if, switch, case, default
```

# Flow Controls

The basic algorithm (and program) is constituted by the following operations:

- Sequential statements: execute instructions in order.
- Selection: first check if the predetermined condition is satisfied, then execute the corresponding instruction.
- Repetition: repeat the execution of some instructions until the criterion fails.



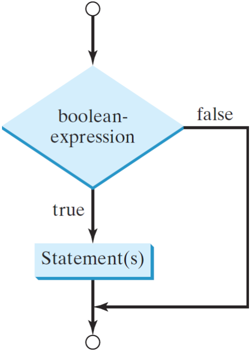

- Note that they are involved with each other generally.
- For example, recall how to find the maximum in the input list?

# Selections

- One-way if statements
- Two-way if-else statements
- Nested if statements
- Multiway if-else if-else statements
- switch-case statements
- Conditional operators

# One-Way if Statements

A one-way if statement executes an action if and only if the condition is true.



```java
...
        if (condition) {
            // selection body
        }
...
```

- The keyword if is followed by the parenthesized condition.
- The condition should be a boolean expression or a boolean value.
- It the condition is true, then the statements in the selection body will be executed once.
- If not, then the program won't enter the selection body and skip the whole selection body.
- Note that the braces can be omitted if the block contains only single statement.

# Example

Write a program which receives a nonnegative number as input for the radius of a circle, and determines the area of the circle.

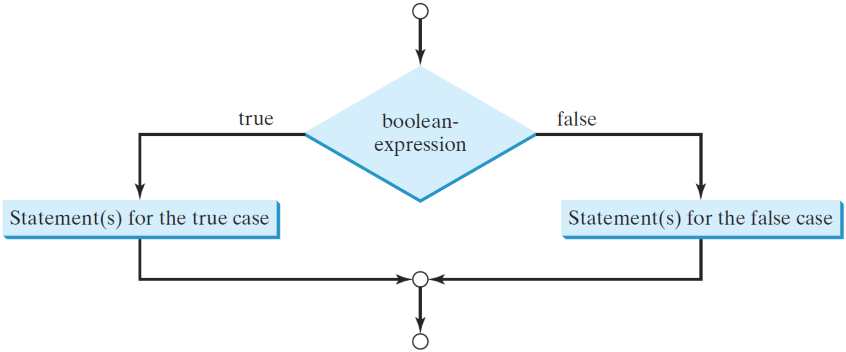
```
...
        double area;
        if (r > 0) {
            area = r * r * 3.14;
            System.out.println(area);
        }
...
```

- However, the world is not well-defined.

# Two-Way if-else Statements

A two-way if-else statement decides which statements to execute based on whether the condition is true or false.

```
...
        if (condition) {
            // body for the true case
        } else {
            // body for the false case
        }
...
```


true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

# Example

Write a program which receives a number as input for the radius of a circle. If the number is nonnegative, then determine the area of the circle; otherwise, output "Not a circle."

```
...
        double area;
        if (r > 0) {
            area = r * r * 3.14;
            System.out.println(area);
        } else {
            System.out.println("Not a circle.");
        }
        input.close();
    }
...
```

# Nested if Statements

- For example,

```
...
        if (score >= 90)
            System.out.println("A");
        else {
            if (score >= 80)
                System.out.println("B");
            else {
                if (score >= 70)
                    System.out.println("C");
                else {
                    if (score >= 60)
                        System.out.println("D");
                    else
                        System.out.println("F");
                }
            }
        }
...
```

# Multi-Way if-else

- Let's redo the previous problem.

```java
...
        if (score >= 90)
            System.out.println("A");
        else if (score >= 80)
            System.out.println("B");
        else if (score >= 70)
            System.out.println("C");
        else if (score >= 60)
            System.out.println("D");
        else
            System.out.println("F");
...
```

- An if-elseif-else statement is a preferred format for multiple alternatives, in order to avoid deep indentation and make the program easy to read.

- The order of conditions may be relevant. (Why?)

```java
...
        if ((score >= 90) && (score <= 100))
        else if ((score >= 80) && (score < 90))
        ...
        else
...
```

- The performance may degrade due to the order of conditions. (Why?)

# Common Errors

```
...
        double area;
        if (r > 0);
            area = r * r * 3.14;
            System.out.println(area);
...
```

# Example

**Generating random numbers**

Write a program which generates 2 random integers and asks the user to answer the math expression.

- For example, the program shows $2 + 5 =?$
- If the user answers 7, then the program reports "Correct." and terminates.
- Otherwise, the program reports "Wrong answer. The correct answer is 7." for this case.
- You may use **Math.random**() for a random value between 0.0 and 1.0, excluding themselves.

```java
...
        int x = (int) (Math.random() * 10); // integers 0 ~ 9
        int y = (int) (Math.random() * 10);
        int answer = x + y;

        System.out.println(x + " + " + y + " =  ?");

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

        if (z == answer)
            System.out.println("Correct.");
        else
            System.out.println("Wrong. Answer: " + answer);
        input.close();
...
```

- Can you extend this program for all arithmetic expressions (i.e., $+ - \times \div$)?

# Exercise

**Find Max**

Write a program which determines the maximum value in 3 random integers whose range from 0 to 99.

- How many variables do we need?
- How to compare?
- How to keep the maximum value?

```java
...
        int x = (int) (Math.random() * 100);
        int y = (int) (Math.random() * 100);
        int z = (int) (Math.random() * 100);

        int max = x;
        if (y > max) max = y;
        if (z > max) max = z;
        System.out.println("max = " + max);
...
```

- In this case, a scalar variable is not convenient. (Why?)
- So we need arrays and loops.

# switch-case Statements

A switch-case structure takes actions depending on the target variable.

```
...
        switch (target) {
            case v1:
                // statements
                break;
            case v2:
            .
            .
            case vk:
                // statements
                break;
            default:
                // statements
        }
...
```

- A switch-case statement is more convenient than an if statement for multiple discrete conditions.
- The variable *target*, always enclosed in parentheses, must yield a value of char, byte, short, int, or **String** type.
- The value $v_1, \ldots,$ and $v_k$ must have the same data type as the variable *target*.
- In each case, a break statement is a must.[29]
    - break is used to break a construct!
- The default case, which is optional, can be used to perform actions when none of the specified cases matches *target*.
    - Counterpart to else statements.

[29] If not, there will be a fall-through behavior.

# Example

```java
...
        // RED: 0
        // YELLOW: 1
        // GREEN: 2
        int trafficLight = (int) (Math.random() * 3);
        switch (trafficLight) {
            case 0:
                System.out.println("Stop!!!");
                break;
            case 1:
                System.out.println("Slow down!!");
                break;
            case 2:
                System.out.println("Go!");
        }
...
```

# Conditional Operators

A conditional expression evaluates an expression based on the specified condition and returns a value accordingly.

```
...
        someVar = booleanExpr ? exprA : exprB;
...
```

- This is the only ternary operator in Java.
- If the boolean expression is evaluated true, then return expr A; otherwise, expr B.

- For example,

```java
...
        if (num1 > num2)
            max = num1;
        else
            max = num2;
...
```

- Alternatively, one can use a conditional expression like this:

```java
...
        max = (num1 > num2) ? num1 : num2;
...
```

```
class Lecture4 {

                    "Loops"

}

// keywords:
while, do, for, break, continue
```

# Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

- For example, a program outputs "Hello, Java." for 100 times.

```
...
        System.out.println("Hello, Java.");
        System.out.println("Hello, Java.");
        .
        . // copy and paste for 100 times
        .
        System.out.println("Hello, Java.");
...
```

```java
...
        int cnt = 0;
        while (cnt < 100) {
            System.out.println("Hello, Java.");
            cnt++;
        }
...
```

- This is a simple example to show the power of loops.
- In practice, any routine which repeats couples of times[30] can be done by folding them into a loop.

[30] I prefer to call them "patterns."

# 成也迴圈，敗也迴圈。

- Loops provide substantial computational power.
- Loops bring an efficient way of programming.
- Loops could consume a lot of time.[31]

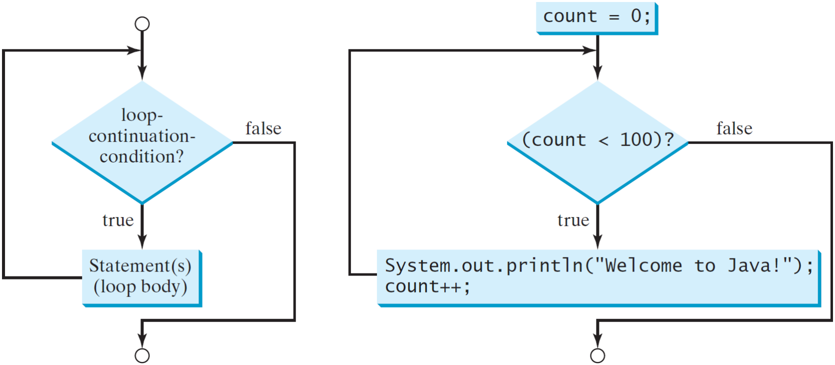
[31] We will visit the analysis of algorithms in the end of this lecture.

# while Loops

A while loop executes statements repeatedly while the condition is true.

```
...
        while (condition) {
            // loop body
        }
...
```

- The condition should be a boolean expression which determines whether or not the execution of the body occurs.
- If true, the loop body is executed and check the condition again.
- Otherwise, the entire loop terminates.



loop-
continuation-
condition?
false
true
Statement(s)
(loop body)
count = 0;
(count < 100)?
false
true
System.out.println("Welcome to Java!");
count++;

# Example

Write a program which sums up all integers from 1 to 100.

- In math, the question can be written as:

$$\text{sum} = 1 + 2 + \cdots + 100.$$

- But this form is not doable in the machine.[32]

[32] We need to develop computational thinking. Read http://rsta.royalsocietypublishing.org/content/366/1881/3717.full or http://blog.orangeapple.tw/posts/what-is-computational-thinking/.

- Normally, the machine executes the instructions sequentially.
- So one needs to decompose the math equation into several steps, like:

```
...
        int sum = 0;
        sum = sum + 1;
        sum = sum + 2;
        .
        .
        .
        sum = sum + 100;
...
```

- It is obvious that many similar statements can be found.

- Using a while loop, the program can be rearranged as follows:

```
...
        int sum = 0;
        int i = 1;
        while (i <= 100) {
            sum = sum + i;
            ++i;
        }
...
```

- You should guarantee that the loop will terminate as expected.
- In practice, the number of loop steps (iterations) is unknown until the input data is given.

# Malfunctioned Loops

- It is really easy to make an infinite loop.

```
...
        while (true);
...
```

- The common errors of the loops are:
  - never start
  - never stop
  - not complete
  - exceed the expected number of iterations

# Example

Write a program which asks the sum of two random integers and lets the user repeatedly enter a new answer until correct.

```
...
        Scanner input = new Scanner(System.in);
        int x = (int) (Math.random() * 10);
        int y = (int) (Math.random() * 10);
        int ans = x + y;

        System.out.println(x + " + " + y + " = ? ");
        int z = input.nextInt();

        while (z != ans) {
            System.out.println("Try again? ");
            z = input.nextInt();
        }
        System.out.println("Correct.");
        input.close();
...
```

# Loop Design Strategy

- Writing a correct loop is not an easy task for novice programmers.
- Consider 3 steps when writing a loop:
  - Find the pattern: identify the statements that need to be repeated.
  - Wrap by loops: put these statements in the loop.
  - Set the continuation condition: translate the criterion from the real world problem into computational conditions.[33]

[33] Not unique.

# Sentinel-Controlled Loops

Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a sentinel value, signifies the end of the loop.
- For example, the operating systems and the GUI apps.

# Example: Cashier Problem

Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

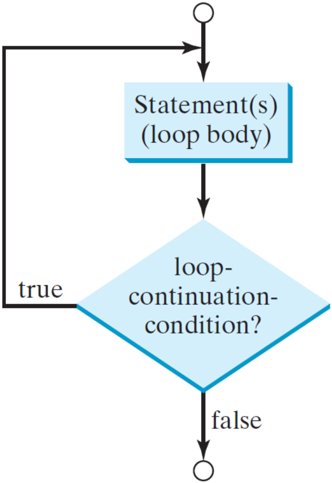
```
1  ...
2          int total = 0, price;
3          Scanner input = new Scanner(System.in);
4
5          System.out.println("Enter price?");
6          price  = input.nextInt();
7
8          while (price > 0) {
9              total += price;
10             System.out.println("Enter price?");
11             price = input.nextInt(); // repeat Line 5 and 6?!
12         }
13
14         System.out.println("Total = " + total);
15         input.close();
16 ...
```

# do-while Loops

A do-while loop is similar to a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```
...
        do {
            // loop body
        } while (condition); // Do not miss the semicolon!
...
```

- Note that there is a semicolon at the end of the do-while loop.
- The do-while loops are also called posttest loops, in contrast to while loops, which are pretest loops.


Statement(s)
(loop body)
loop-
continuation-
condition?
true
false

# Example (Revisted)

Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```
...
        int total = 0, price = 0;
        Scanner input = new Scanner(System.in);

        do {
            total += price;
            System.out.println("Enter price?");
            price = input.nextInt();
        } while (price > 0);

        System.out.println("Total = " + total);
        input.close();
...
```

# for Loops

A for loop generally uses a variable to control how many times the loop body is executed.

```
...
        for (init_action; condition; increment) {
            // loop body
        }
...
```

- *init-action*: declare and initialize a variable
- *condition*: set a criterion for loop continuation
- *increment*: how the variable changes after each iteration
- Note that these terms are separated by semicolons.

# Example

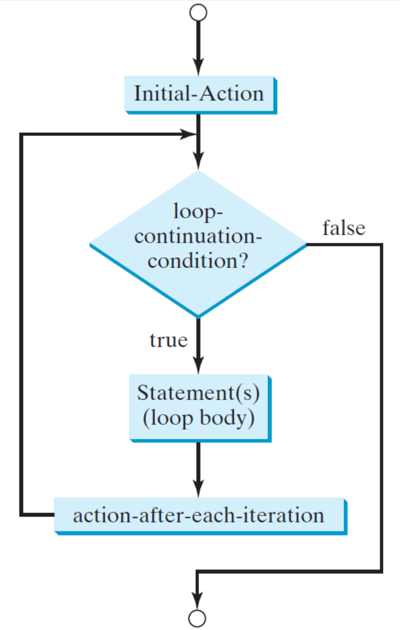
## Sum from 1 to 100

Write a program which sums from 1 to 100.

```
...
        int sum = 0;
        for (int i = 1; i <= 100; ++i)
            sum = sum + i;
...
```

- Compared to the while version,

```
...
        int sum = 0;
        int i = 1;
        while (i <= 100) {
            sum = sum + i;
            ++i;
        }
...
```


Initial-Action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration

# Example: Selection Resided in Loop

**Display all even numbers**

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where k is an integer.

- You may use the modular operator (%).

```
...
        for (int i = 1; i <= 100; i++) {
            if (i % 2 == 0) System.out.println(i);
        }
...
```

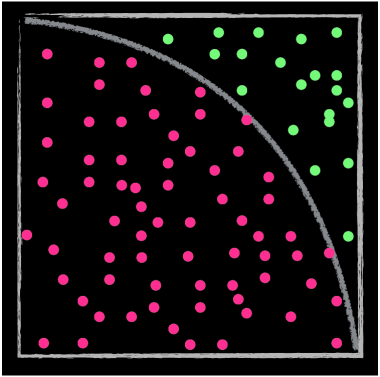
- Also consider this alternative:

```
...
        for (int i = 2; i <= 100; i += 2) {
            System.out.println(i);
        }
...
```

- How about odd numbers?

# Example: Monte Carlo Simulation[34]

- Write a program which conducts a Monte Carlo simulation to estimate $\pi$.



[34] See https://en.wikipedia.org/wiki/Monte_Carlo_method.