# Lecture 2
# Elementary Programming



## Writing a simple program

- Writing a program involves
  - Designing *algorithms*, and
  - Translating algorithms into programming instructions, or code
- Algorithm:
  - describes how a problem is solved by listing the actions that need to be taken and the order of their execution
  - help the programmer plan a program before writing it in a programming language
  - can be described in natural languages or in *pseudocode*
- Pseudocode:
  - natural language mixed with some programming code



## Example

- Calculate the area of a circle
- Algorithm:

1. Read in the circle's radius.
2. Compute the area using the following formula:

$$area = radius \times radius \times \pi$$

3. Display the result.



## Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
  double radius;
  double area;

  // Assign a radius
  radius = 20;

  // Compute area
  area = radius * radius * 3.14159;

  // Display results
  System.out.println("The area for the circle of
   radius " + radius + " is " + area);
 }
}
```

allocate memory for radius

radius: no value

# Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of
    radius " + radius + " is " + area);
 }
}
```

memory

| | |
|---|---|
| radius | no value |
| area | no value |

allocate memory for area



# Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of
    radius " + radius + " is " + area);
 }
}
```

assign 20 to radius

| | |
|---|---|
| radius | 20 |
| area | no value |



# Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of
    radius " + radius + " is " + area);
 }
}
```

memory

| | |
|---|---|
| radius | 20 |
| area | 1256.636 |

compute area and assign it to variable area



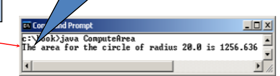
# Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of
    radius " + radius + " is " + area);
 }
}
```

memory

| | |
|---|---|
| radius | 20 |
| area | 1256.636 |

print a message to the console



## Reading Input from the Console

1. Create a Scanner object

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

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value. For example,

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



## 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.



## Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " +
  area + " for radius "+radius);
```



## Declaring Variables

```
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;
```

## Assignment Statements

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



## Declaring and Initializing in One Step

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



## Constants

Syntax:

```
final datatype CONSTANTNAME = VALUE;
```

Example:

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



## Numerical Data Types

| Name | Range | Storage Size |
|---|---|---|
| byte | $-2^{7}$ (-128) to $2^{7}-1$ (127) | 8-bit signed |
| short | $-2^{15}$ (-32768) to $2^{15}-1$ (32767) | 16-bit signed |
| int | $-2^{31}$ (-2147483648) to $2^{31}-1$ (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 |

## 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 |



## Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)



## NOTE

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

  – displays 0.5000000000000001, not 0.5, and

```
System.out.println(1.0 - 0.9);
```

  – displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.



## Number Literals

- A *literal* is a constant value that appears directly in the program.
- For example, 34, 1,000,000, 5.0, and *true* are literals in the following statements:

  – `int i = 34;`
  – `long x = 1000000;`
  – `double d = 5.0;`
  – `Boolean b = true;`

## Integer Literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable.
- A compilation error would occur if the literal were too large for the variable to hold.
- For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.
- An integer literal is assumed to be of the int type, whose value is between $-2^{31}$ to $2^{31}–1$.
- To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).



## Floating-Point Literals

- Floating-point literals are written with a decimal point.
- By default, a floating-point literal is treated as a double type value.
- For example, 5.0 is considered a double value, not a float value.
- You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D.
- For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.



## Scientific Notation

- Floating-point literals can also be specified in scientific notation.
- For example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456.
- E (or e) represents an exponent and it can be either in lowercase or uppercase.



## Arithmetic Expressions

$$\boxed{\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9(\frac{4}{x}+\frac{9+x}{y})}$$

is translated to

(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)

## How to Evaluate an Expression

```
3 + 4 * 4 + 5 * (4 + 3) - 1
```
(1) inside parentheses first
```
3 + 4 * 4 + 5 * 7 - 1
```
(2) multiplication
```
3 + 16 + 5 * 7 - 1
```
(3) multiplication
```
3 + 16 + 35 - 1
```
(4) addition
```
19 + 35 - 1
```
(5) addition
```
54 - 1
```
(6) subtraction
```
53
```



## Shortcut Assignment Operators

| *Operator* | *Example* | *Equivalent* |
|---|---|---|
| `+=` | `i += 8` | `i = i + 8` |
| `-=` | `f -= 8.0` | `f = f - 8.0` |
| `*=` | `i *= 8` | `i = i * 8` |
| `/=` | `i /= 8` | `i = i / 8` |
| `%=` | `i %= 8` | `i = i % 8` |



## Increment and Decrement Operators

| **Operator** | **Name** | **Description** |
|---|---|---|
| ++var | preincrement | The expression (++var) increments var by 1 and evaluates to the *new* value in var *after* the increment. |
| var++ | postincrement | The expression (var++) evaluates to the *original* value in var and increments var by 1. |
| --var | predecrement | The expression (--var) decrements var by 1 and evaluates to the *new* value in var *after* the decrement. |
| var-- | postdecrement | The expression (var--) evaluates to the *original* value in var and decrements var by 1. |



## Increment and Decrement Operators

```
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;
```

## Increment and Decrement Operators

- 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.



## Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```



## Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.



## Type Casting

- 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)`

What is wrong? int x = 5 / 2.0;

```
                range increases
         ---------------------------->
byte, short, int, long, float, double
```

## Escape Sequences for Special Characters

| *Description* | *Escape Sequence* | *Unicode* |
|---|---|---|
| Backspace | `\b` | `\u0008` |
| Tab | `\t` | `\u0009` |
| Linefeed | `\n` | `\u000A` |
| Carriage return | `\r` | `\u000D` |
| Backslash | `\\` | `\u005C` |
| Single Quote | `\'` | `\u0027` |
| Double Quote | `\"` | `\u0022` |



## Casting between char and Numeric Types

```
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```



## Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.



## Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 1156

remainingAmount initialized

# Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 1156

numberOfOneDollars 11

numberOfOneDollars assigned



# Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 56

numberOfOneDollars 11

remainingAmount updated



# Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 56

numberOfOneDollars 11

numberOfOneQuarters 2

numberOfOneQuarters assigned



# Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 6

numberOfOneDollars 11

numberOfQuarters 2

remainingAmount updated

## Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles



## 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.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
  - For example, the variables `radius` and `area`, and the method `computeArea`.



## Naming Conventions, cont.

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and` MAX_VALUE

## Proper Indentation and Spacing

- Indentation
  - Indent two spaces.
- Spacing
  - Use blank line to separate segments of the code.



## Block Styles

Use *end-of-line style* for braces.

*Next-line style*

```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

*End-of-line style*

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```