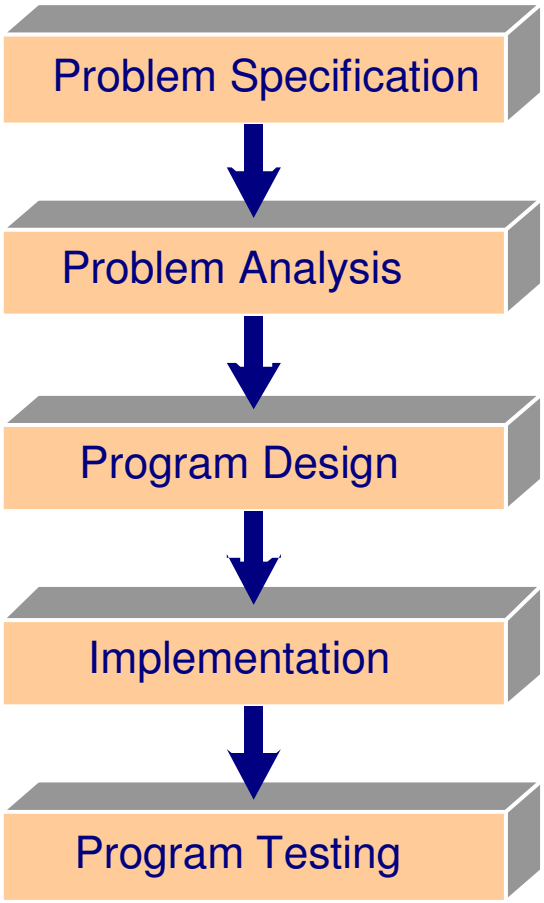
Chapters 3 & 5

Data and Operators

Review: Java Program Development

- Aim: for Problem Solving
- Program Development Process

1) Problem Specification
   - identify user requirements
2) Problem Analysis
   - identify inputs, outputs, formulas
3) Program Design
   - write solution steps
   (using pseudocode or flowcharts)
   - go through dry-run to test the
   solution steps using test samples
4) Implementation
   - translate solution into computer program
5) Program Testing
   - use test samples to test the program



Data and Operators

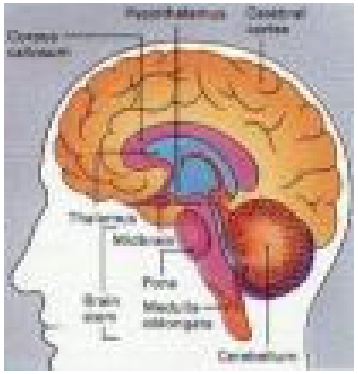
- **Computer Memory**
- Data Types
- Literals
- Identifiers
- Constants
- Variables

Data

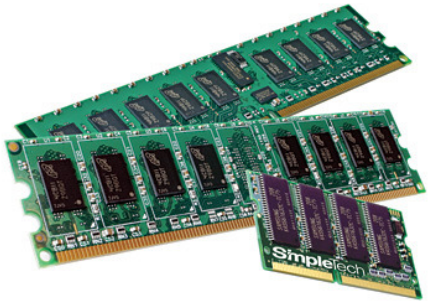
- Fundamental Arithmetic Operators
- Data Type Conversion

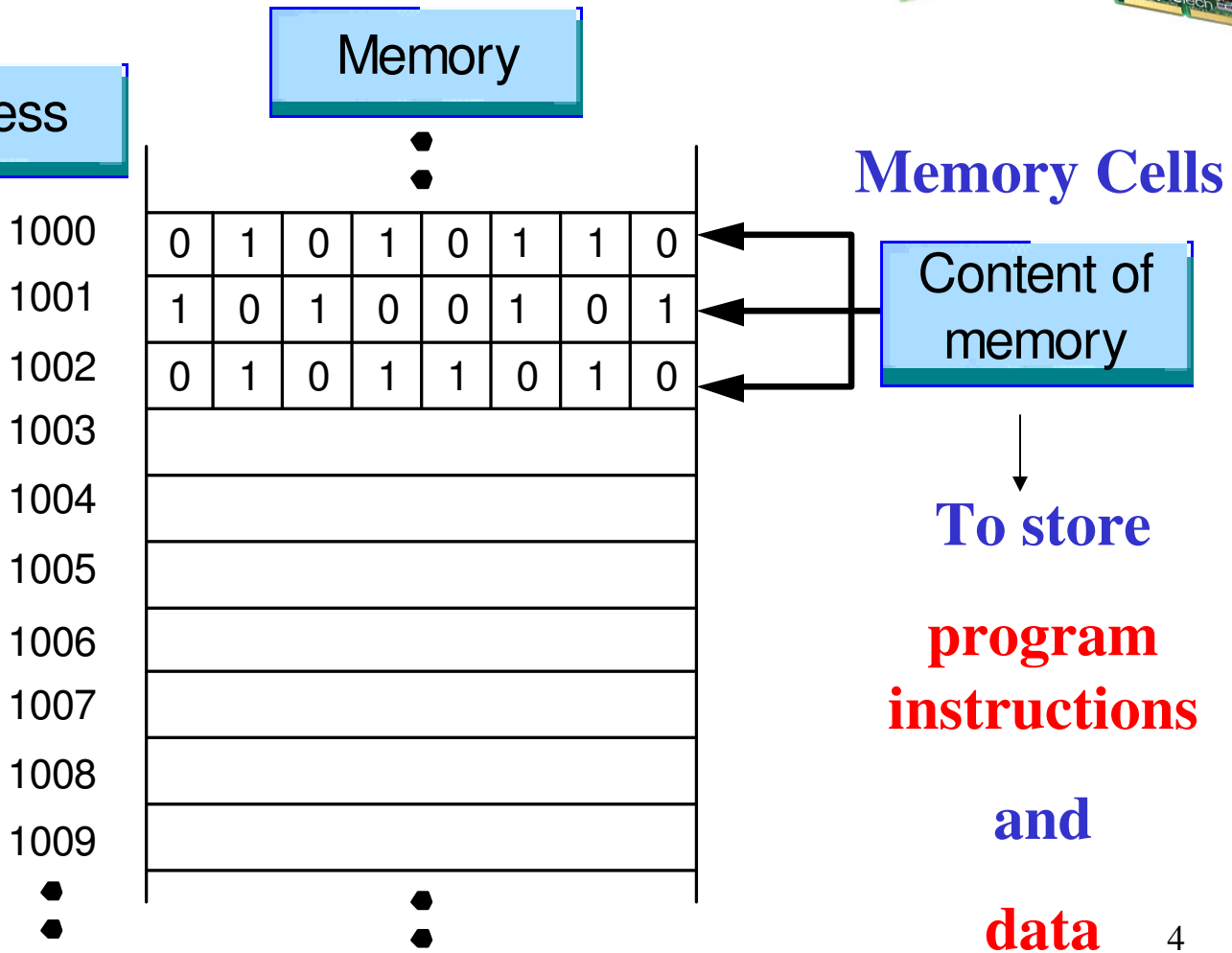
Operators

- Programming Style
- Case Study



Computer Memory



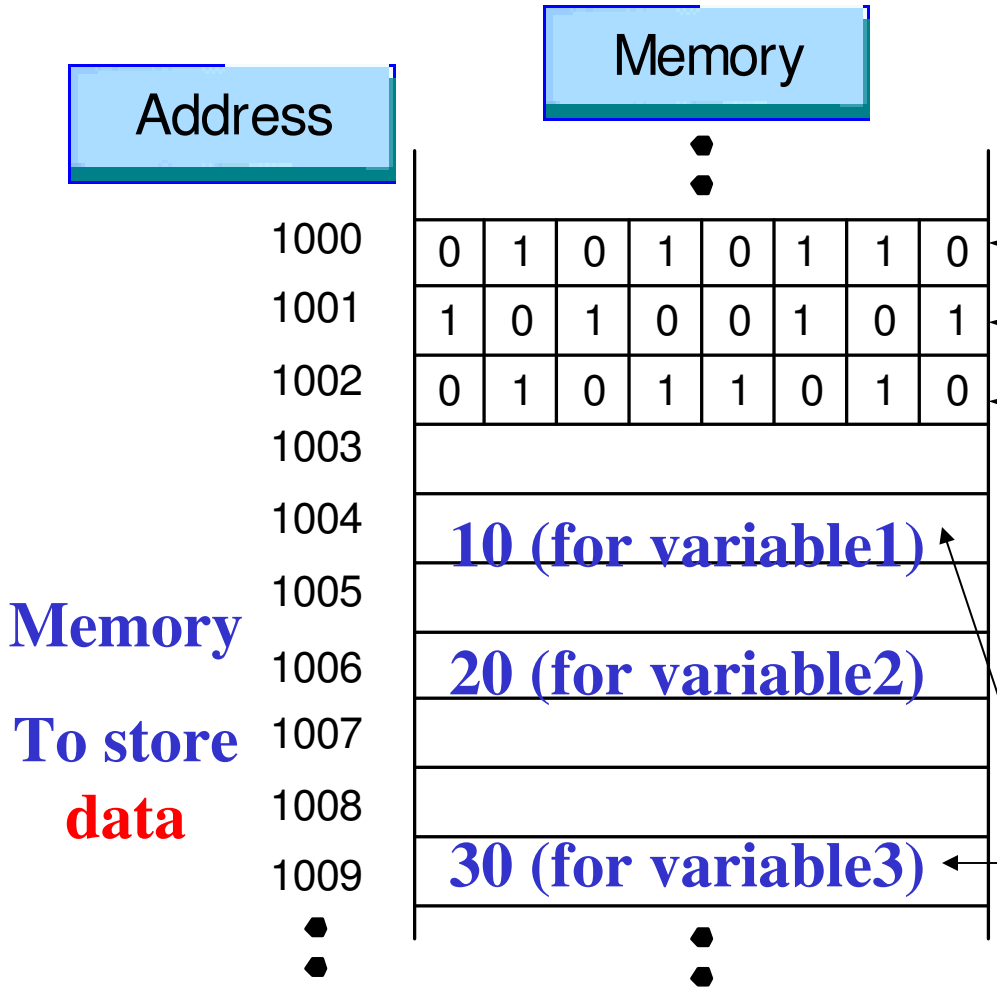


Sequential Storage

8 bits = 1 byte (binary format)

Memory and Variables

for data objects



Program logic

DataType	Variable	Value

```
int variable1=10;
int variable2=20;
int variable3;
variable3 = variable1 + variable2;
....
```

Data and Operators

- Computer Memory
- **Data Types**
- Literals
- Identifiers
- Constants
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- Case Study

Data Types

- A program needs to work with **data**.
- A **data object** is of a certain **type** that requires the specified **memory storage size**. The data types in Java are
  - Primitive Data Types
  - Reference Data Types

Why Variables & Data Type?

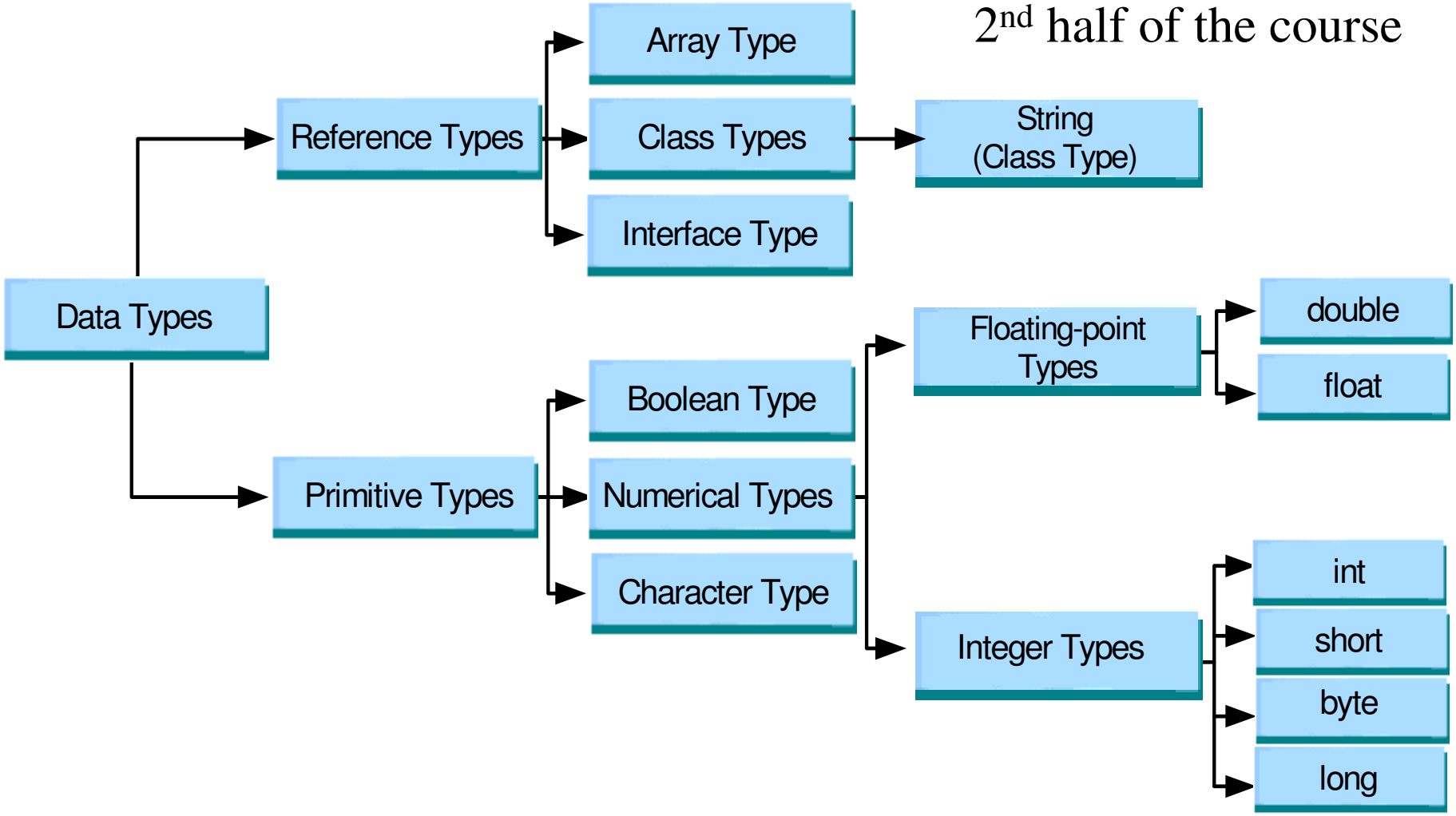
- A **data object** in a Java program can be
  - a **constant**
  - a **variable**

} **Declared with Data Type**

Java reserves the necessary data **storage locations** depending on the data type of the data object.

Data Types

2nd half of the course



Primitive Data Types

- Four integer number data types:
  - byte, short, int, long
- Two floating-point number data types:
  - float, double
- Character data type:
  - char
- Boolean data type:
  - boolean (true or false)

(1) Integer Data Types

* There is a zero in the middle

Data Type	Storage Size	Range
`byte`	1 byte	`-128 to 127`
`short`	2 bytes	`-32,768 to 32,767`
`int`	4 bytes	`-2,147,483,648 to 2,147,483,647`
`long`	8 bytes	`-9,223,372,036,854,775,807 to 9,223,372,036,854,775,808`

(2) Floating-point Data Types

Data Type	Storage Size	Range	Precision
`float`	4 bytes	`-3.4e38` `to` `3.4e38`	6 to 7 significant digits
`double`	8 bytes	`-1.7e308` `to` `1.7e308`	14 to 15 significant digits

* Precision (may not be precise) / * significant digits

http://www.psc.edu/general/software/packages/ieee/ieee.php

(3) Character and (4) Boolean Data Types

Character Data Type

- Any data object which is
  - an English letter (a,..,z,A,..,Z)
  - an English punctuation mark (!, ?, etc.)
  - a decimal digit (0,..,9)
  - a symbol such as a space
  - etc…
- Storage space - **2 bytes**

Boolean Data Type

- boolean literals: true or false
- Storage space - 1 byte

Reference Data Types
String Type

- A **reference data type** is used to store an **address** of an object so that we can use the address to **refer** to the object data.

- **String Data Type**

  – Strings are represented by a sequence of characters within double quotes,

    e.g. “Hui Siu Cheung”, “Hello Students!”

  – Strings are **not** primitive data types

  – Java provides the String class with methods

Data Type Quiz

Chapter 3
Data Types, Constants and Variables

- Computer Memory
- Data Types
- **Literals**
- Identifiers
- Constants
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- Case Study

Literals

- Literals are **values** (associated with data type) used in the program.

- **Five types of literals:**
  - Integer literals, e.g. 100, –256
  - Floating-point literals, e.g. 2.4, –3.0
  - Character literals, e.g. 'a', '+'
  - Boolean literals, e.g. **true**, **false**
  - String literals, e.g. **"Hello Students"**

(1) Integer Literals

- With type **int** (default).
- To denote integer literal of the long type, append the letter **L** or **l** to it (e.g. 12345678L).
- To denote an octal integer literal, add a leading zero, 0 to it.
- To denote a hexadecimal integer literal, add a leading 0x or 0X (zero x) to it. E.g. 0xFFFF.

(2) Floating-point Literals

- With type **double** (default).
- To make a number a float type, append the letter **f** or **F** (e.g. 10.34f).
- You may also append the letter d or D to make a the number double (e.g. 10.34d).
- Can use Scientific notations:
  e.g. 1.23456e2 (i.e. $1.23456 \times 10^{2} = 123.456$)
  1.23456e-2 (i.e. $1.23456 \times 10^{-2} = 0.0123456$)

(3) Character Literals

- A character is stored as **16-bit** (**2 bytes**) unsigned values using **Unicode** encoding scheme which is established by the Unicode Consortium to support interchange, processing, and display of text in different languages such as Chinese and Korean.

- Unicode includes the **ASCII** (American Standard Code of Information Interchange) code. A total of 128 characters. Each character is enclosed with **quotes**, e.g. 'A'. (single quote)

- Each character in the Unicode has a corresponding **numeric code**, e.g. 'A' has a value of 41 (or '\u0041' in Unicode (2 bytes)) in hexadecimal representation (or 65 in decimal). (See **Appendix A** of the textbook)

ASCII Unicode Set (1 byte)

	0	1	2	3	4	5	6	7	8	9
0	NUL							BEL	BS	TAB
1	LF		FF	CR						
2								ESC		
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Examples of Escape Sequence

- Some useful **non-printable control characters** are referred to by the ***escape sequence*** which is a better alternative, in terms of memorization, than numbers. e.g. '\n' the newline (or linefeed) character instead of the number **10**.

* Escape for printing a special character / * Formatting

'\a'	**alarm bell**	'\f'	**form feed**	'\n'	**newline**
'\t'	**horizontal tab**	'\"'	**double quote**	'\v'	**vertical tab**
'\b'	**back space**	'\\'	**backslash**	'\r'	**carriage return**
'\''	**single quote**				

Chapter 3
Data Types, Constants and Variables

- Computer Memory
- Data Types
- Literals
- **Identifiers**
- Constants
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- Case Study

Identifiers

Identifiers are used to **name** things such as **variables**, **constants**, **classes** and **packages**.

Rules for naming identifiers:

1. An identifier must start with a letter, an underscore character (_) or a dollar sign ($).
2. An identifier may contain only letters, digits (0,..,9), and the underscore character (_).
3. An identifier is case sensitive.
4. An identifier cannot contain a space, or any other characters such as a dot (.) or an asterisk (*).
5. An identifier cannot be a reserved word or keyword.
6. An identifier does not have any length limit.

Example: **Valid** - area, Length, a, A, sevenAnd1, my_program
Invalid - new, my-program, Java.com, 7and1

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- **Constants**
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- Case Study

Constants

- A constant is an object whose value is **unchanged** throughout program execution.
- Define a constant by using the Java keywords **final**. The form is

```
final Type CONSTANT_NAME = Value;
```

where CONSTANT_NAME - name of the constant.
(where CONSTANT_NAME should use *upper* case, separated with an underscore if comprising two or more words)

e.g. `final double PI = 3.14159;`

- If you place the constant outside the main() method, you need to use: `static final PI = 3.14159 ;`
- During compilation, the value of the constant will be **substituted** whenever the name of the constant appears in the program.

Constants

By giving a **symbolic name** to a constant:
- it improves the **readability** of the program
- it makes programs easier to be **modified**

Example:

```
public class DefineConstant {
  static final double PI = 3.14159;
   public static void main(String[] args) {
      System.out.println("The value for pi is " + PI);
   }
}
```

Output:

The value for pi is 3.14159

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- Constants
- **Variables**
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- Case Study

Variables

- **Variables** are data objects that may **change** and be **assigned values** as the program runs.
- Each **variable** has a **name**. The name of a variable is a sequence of alphanumeric characters plus the underscore _. The first character must be a letter or an underscore. Variable names are case sensitive.

Dollar sign ($)???

- Use **meaningful** names
  - making programs more readable. e.g. 'tax' is easier to understand than 't'.

Example:
- **Valid**: goodName, newValue, taxRate, etc.
- **Invalid**: good Name, new-Value, tax+Rate, 7And11

- A variable name cannot be any of the ***keywords*** in Java. Keywords have special meanings to Java compiler.

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

- Each variable has a **data type** such as *int*, *float* and *char*.

- Variables are declared by ***declaration statements***. A declaration can be done with or without initialization.

- A declaration statement without initialization has the form

Type Variable[, Variable];

where Type can be int, float, or char, etc.
Variable is the name of the variable.
[...] may be repeated zero or more times.

Program: Variables without Initialization

Specification: Computing Income Tax

CONSTANTS

```
public class DeclareVariable {
   static final int DEDUCTION = 2000;     // Constant
   static final double TAX_RATE = 0.2;    // Constant
   public static void main(String[ ] args){
```

VARIABLES

```
      // Define Variables
      double incomeTax, taxableIncome, grossSalary;
      int numOfDependents, numOfChildren, numOfParents;

      // Assign or Read numOfChildren, numOfParents, grossSalary

      // Compute  income tax

      // Print the income tax

   }
}
```

LOGIC IN SEQUENCE (Note the order of statements - dependent)

Program: Computing Income Tax

```
public class DeclareVariable {
   static final int DEDUCTION = 2000;     // Constant
   static final double TAX_RATE = 0.2;    // Constant
   public static void main(String[ ] args){
      // Variables
      double incomeTax, taxableIncome, grossSalary;
      int numOfDependents, numOfChildren, numOfParents;

      // Assignment statements
      numOfChildren = 2;
      numOfParents = 2;
      grossSalary = 100000.0;

      numOfDependents = numOfChildren + numOfParents;
      taxableIncome =
          grossSalary – numOfDependents*DEDUCTION;
      incomeTax = taxableIncome * TAX_RATE;

      System.out.println("The income tax is " + incomeTax);
   }
}
```

LOGIC IN SEQUENCE

USE ASSIGNMENT STATEMENTS

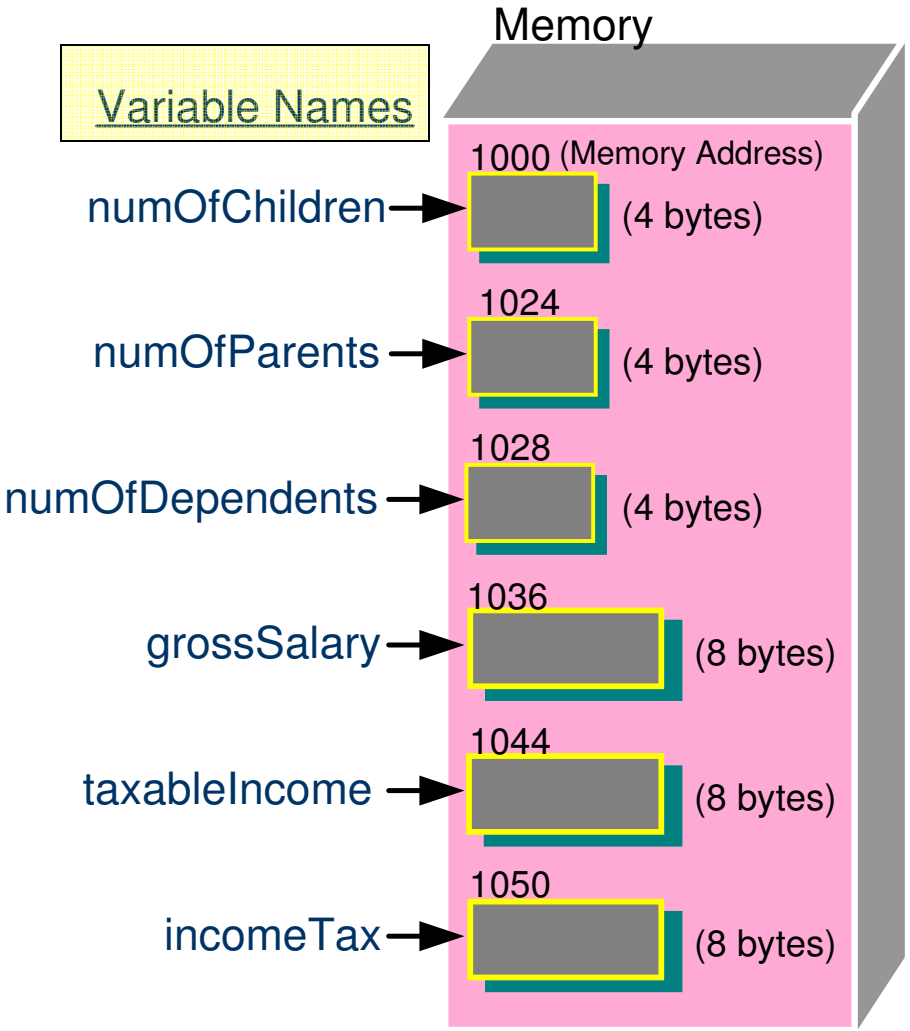
READ INPUT

COMPUTE

PRINT OUTPUT

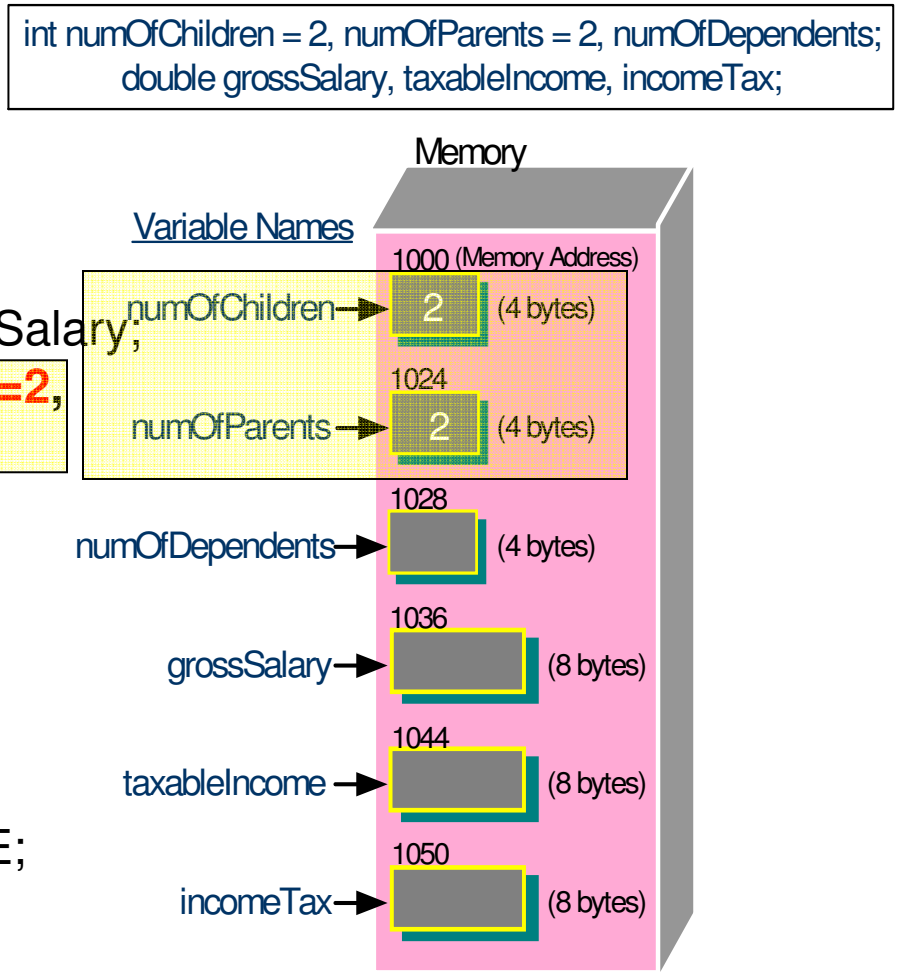
```
Program output:
The income tax is 18400.0
```

```
int numOfChildren, numOfParents, numOfDependents;
double grossSalary, taxableIncome, incomeTax;
```



Program: Variables with Initialization

```
public class DeclareVarInit {
    // constants
    static final int DEDUCTION = 2000;
    static final double TAX_RATE = 0.2;
    public static void main(String[] args) {
        // variables
        double incomeTax, taxableIncome, grossSalary;
        int numOfDependents, numOfChildren=2,
            numOfParents=2;
         // assignment statements
        grossSalary = 100000;
        numOfDependents = numOfChildren +
            numOfParents;
        taxableIncome = grossSalary –
            numOfDependents*DEDUCTION;
        incomeTax = taxableIncome * TAX_RATE;
        System.out.println("The income tax is " +
            incomeTax);
    }
}
```



- **Initialization** can also be done as part of a declaration. This means a variable is given a starting value when it is declared.

- To improve readability of the program - declare initialized and un-initialized variables in separate declaration statements.
  Example:

```
int numOfChildren=2, numOfParents = 2;
int numOfDependents;
```

- During compilation, a **memory location** of suitable size is assigned for each variable. Naturally a variable must be declared before it is used.

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- Constants
- Variables
- **Fundamental Arithmetic Operators**
- Data Type Conversion
- Programming Style
- Case Study

Operators

An **operator** is a symbol that causes some operations to be performed on one or more variables (and data values: literals and constants).

Java Operators:

- fundamental arithmetic operators
- assignment operators
- increment/decrement operators
- arithmetic assignment operators
- concatenation operators
- relational operators
- logical operators

In this chapter

In chapter 6

1. Fundamental Arithmetic Operators

- **unary** operators: can change the sign of a value

  +, - : e.g. +31, -5

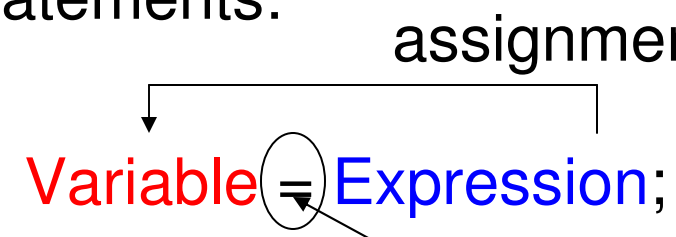
- **binary** operators:

+ :	addition	e.g. 7+5=12
- :	subtraction	e.g. 7-5=2
* :	multiplication	e.g. 7*5=35
/ :	division	e.g. 7/5=1
% :	modulus	e.g. 7%5=2

Note: integer division returns integer results.

2. Assignment Operators

- An assignment statement is a statement to assign a value to a variable.
- The value of a variable may be changed by the following form of statements:



The variable Variable is assigned the value of the Expression.
The symbol = is called assignment operator.

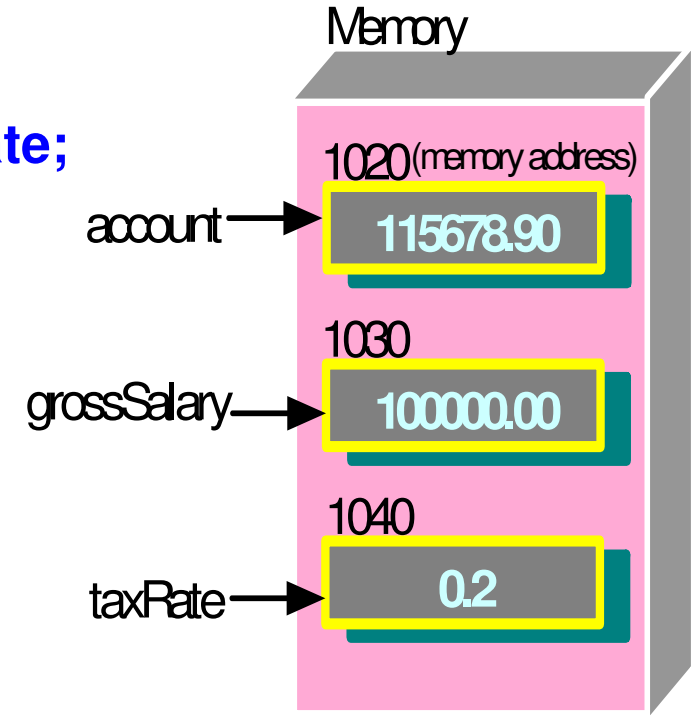
Note: the left hand side must be a **variable name** and not a constant or expression with operators.

Example: Computing new balance

```
public class AssignmentOp {
    public static void main(String[] args)
    {
        double account, grossSalary, taxRate;
        // initialize Variables
        account = 115678.90;
        grossSalary = 100000.00;
        taxRate = 0.2;

        // compute new balance

        // print new balance
    }
}
```



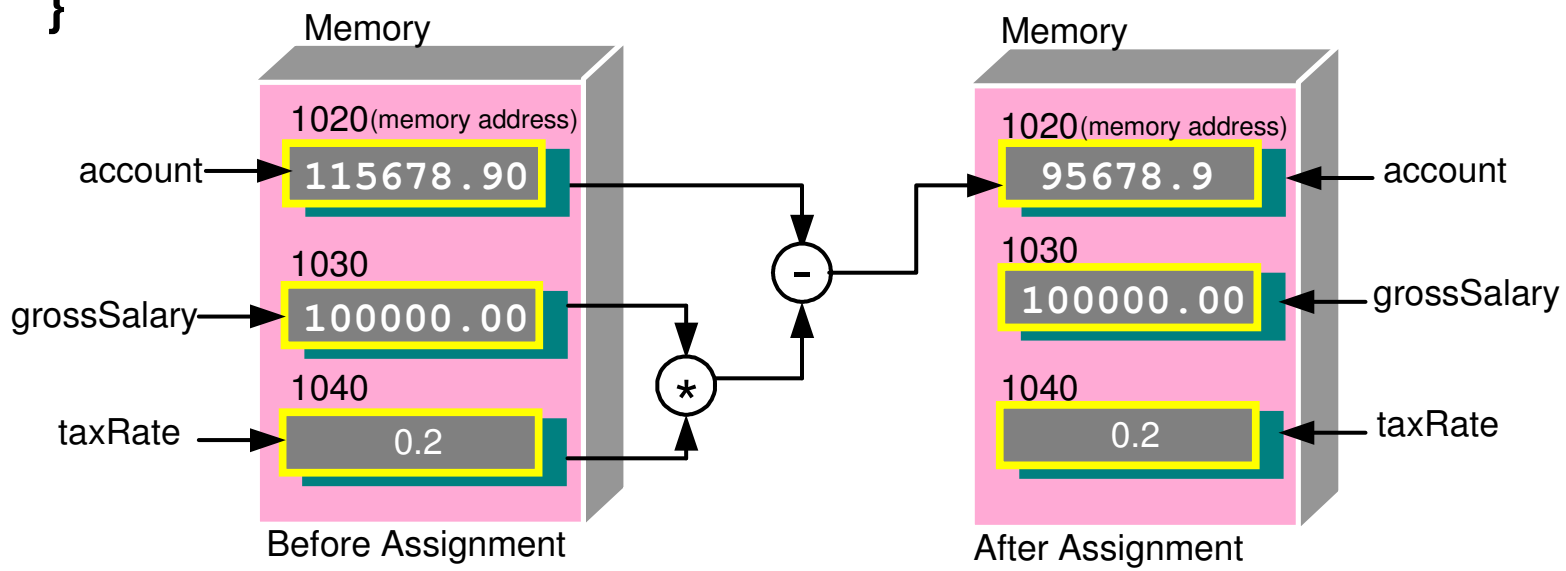
Program Output

```
Your new account balance is 95678.9
```

```
public class AssignmentOp {
    public sttaic void main(String[ ] args)
    {
        double account, grossSalary, taxRate;
        account = 115678.90;
        grossSalary = 100000.00;
        taxRate = 0.2;

        account = account - grossSalary * taxRate;

        System.out.println("Your new account balance is" + account);
    }
}
```



3. Increment/decrement Operators

- **increment operator** : ++ can be used in two ways, prefix and postfix modes. In both forms, the variable will be incremented by 1.
- **In prefix mode: ++Variable.**
  (1) Variable is incremented by 1
  (2) then the value of the expression is the updated value of Variable.
- **In postfix mode: Variable++.**
  (1) The value of the expression is the current value of Variable
  (2) then Variable is incremented by 1.
- The way the **decrement operator** '--' works is the same as the ++, except that the variable is decremented by 1

Example:

```
public class IncDecOp {
   public static void main(String[] args){
      int num1 = 5, num2 = 5;

      System.out.println("num1 = " + num1);
      System.out.println("num1++ = " + num1++);
      System.out.println("num1 = " + num1);
      System.out.println("++num1 = " + ++num1);
      System.out.println("num1 = " + num1);

      System.out.println("num2 = " + num2);
      System.out.println("num2-- = " + num2--);
      System.out.println("num2 = " + num2);
      System.out.println("--num2 = " + --num2);
      System.out.println("num2 = " + num2);
   }
}
```

Program Output

```
num1 = 5
num1++ = 5
num1 = 6
++num1 = 7
num1 = 7

num2 = 5
num2-- = 5
num2 = 4
--num2 = 3
num2 = 3
```

Note: 1) Syntax; 2) Data Type: Numeric/char;
3) side effect!!! 4) Good style about it

4. Arithmetic Assignment Operators

General Syntax

```
Variable op = Expression;
```

where op= is +=, -=, *=, /=, %=

This is equivalent to

```
Variable = Variable op Expression;
```

Variable += Expression; => Variable = Variable + Expression;

Variable -= Expression; => Variable = Variable - Expression;

Variable *= Expression; => Variable = Variable * Expression;

Variable /= Expression; => Variable = Variable / Expression;

Variable %= Expression; => Variable = Variable % Expression;

Expressions

An **expression** produces a value (after evaluate), e.g.:

- A constant or literals, e.g. 2, ‘h’, -6.7
- A variable e.g. income, account
- A combination of operators and operands

  e.g. a + b / c

  t * (g - 5)
- An expression may involve incrementing or decrementing other variables

  e.g. 3 * num--
- An expression may also involve assignments

  e.g. num + (h = 5)

Beware of side effect!!!!

Example: Variable += Expression:

```
public class ArithAssignmentOp {
  public static void main(String[] args) {
   double account = 2000.00, income = 1000.00;
   System.out.println(" account = " +
      account + ", income = " + income);
   account += income;
   System.out.println(" account = " +
      account + ", income = " + income);
  }
}
```

Program Output

```
account = 2000.00, income = 1000.00
account = 3000.00, income = 1000.00
```

Assignment with Increment/Decrement Operators

Observe the operation sequence

```
Variable_1 = Variable_2++;
```

```
(1) Variable_1 = Variable_2;
(2) Variable_2 = Variable_2 + 1;
```

```
Variable_1 = ++Variable_2;
```

```
(1) Variable_2 = Variable_2 + 1;
(2) Variable_1 = Variable_2;
```

```
Variable_1 = Variable_2--;
```

```
(1) Variable_1 = Variable_2;
(2) Variable_2 = Variable_2 - 1;
```

```
Variable_1 = --Variable_2;
```

```
(1) Variable_2 = Variable_2 - 1;
(2) Variable_1 = Variable_2;
```

Using incremental/decremental operators

```
public class AssignIncDecOp {
    public static void main(String[] args) {
        int num = 10, counter = 20;
        System.out.println("num = " + num +
                ", counter = " + counter);
        num++;
        ++counter;
        System.out.println("num = " + num +
                ", counter = " + counter);

        num = 0;
        counter = 10;
        System.out.println("num = " + num +
                ", counter = " + counter);
        num = counter--;
        System.out.println("num = " + num +
                ", counter = " + counter);
        num = --counter;
        System.out.println("num = " + num +
                ", counter = " + counter);
    }
}
```

Program Output

```
num = 10,
counter = 20
num = 11,
counter = 21


num = 0,
counter = 10
num = 10,
counter = 9
num = 8,
counter = 8
```

- Multiple assignments can also be done in one statement:

```
Variable_1= Variable_2= ...= Variable_N = Expression;
```

This is equivalent to

```
Variable_N = Expression;
....
Variable_2 = Expression;
Variable_1 = Expression;
```

5. Concatenation Operators

The concatenation operator **+** is used to join strings together to produce a **new string**. (strings + other data types)

The general syntax for using the print() and println() methods:

```
System.out.println(Output [+ Output]);
```

where Output can be Strings, variables or constants of different data types such as char, int, float and double.

[…] may be repeated zero or more times but need to be separated by the concatenation operator.

Using Concatenation Operator

```
public class ConcatenationOp {
   public static void main(String[] args) {
      String s = "Hello Students!";
      char c = 'A';
      int i = 3;
      float j = 6.8f;
      double k = 9.88888;
      System.out.println("String s = " + s);
      System.out.println("char c = " + c +
            "\nint i = " + i);
      System.out.print("float j = " + j);
      System.out.println("\ndouble k = " + k);
   }
}
```

Program Output

```
String s = Hello Students!
char c = A
int i = 3
float j = 6.8
double k = 9.88888
```

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- Constants
- Variables
- Fundamental Arithmetic Operators
- **Data Type Conversion**
- Programming Style
- Case Study

Data Type Conversion

- Type conversion is the conversion of one data type into another type.
- It is needed when more than one type of data objects appear in an expression/assignment. For example, the following statement adds two numbers with different data types:

```
a=1+3.45;
```

where 1 - *integer* ; 3.45 - *floating point*

⇒ However, the addition can only be done if these two numbers are of the same type.

- **Three types of conversions**:
  1. Explicit conversion
  2. Assignment conversion
  3. Arithmetic conversion

1) Explicit Conversion

- It uses the type cast operators,

```
(Type), i.e. (int),  (float), ..., etc.
```

This converts the operand into the type as indicated by the operator.

For example:

```
int num;
double result;
result = (double)num;
```

2) Arithmetic Conversion

- In any operation that involves operands of two different types. It converts the operands to the type of the higher ranking of the two.
  The ranking of the types from high to low:

double
float
long
int
short
byte

Why this ranking?

For example:

```
double ans1, ans2;
ans1 = 1.23 + 5/4;        // 5/4=1, ans1=2.23
ans2 = 1.23 + 5.0/4;      // 5.0/4=1.25, ans2=2.48
```

3) Assignment Conversion

- It converts the type of the result of computing the expression to that of the type of the left hand side if they are different.

- If the variable on the left-hand side of the assignment statement has a higher rank or same rank as the expression, then there is no loss of information.

- Otherwise, there could be a **loss of information.** This is because the lower ranking type variable may not have enough memory storage to store the value of higher ranking type.

  (Data range!!!)

  For example:

```
int i;
double x=2.5, y=5.3;
i = x+y;            // i will have a value of 7.
```

In fact, Java will give you an error message during compilation.

NB: **floating point** result assigned to an **integer variable**

=> loss of information

Examples: Data Type Conversion

```
public class DataTypeConversion {
    public static void main(String[] args) {
        int num;
        double num1, num2;

        // Explicit conversion
        num = (int) 2.5 + (int) 3.7;
        // convert 2.5 into 2 & 3.7 into 3, then add together
        System.out.println("num = " + num);

        // Assignment conversion
        num1 = 2 + 3;
        // add 2 and 3 to get 5, then convert it to 5.0
        System.out.println("num1 = " + num1);

        // Arithmetic conversion
        num2 = 2 + 3.7;
        // convert 2 to 2.0, then perform addition
        System.out.println("num2 = " + num2);
    }
}
```

Program Output

```
num = 5
num1 = 5
num2 = 5.7
```

Special with char: Conversion between char and Numeric Types

- **Conversion from integer to character** – only lower sixteen bits of data are used. For example,

  char ch = (char)0xAB0041; // ch is 'A'

- **Conversion from floating-point to char** – the integral part of the floating-point value is cast into char. For example,

  char ch = (char)65.78; // ch is 'A'

- **Conversion from char to a numeric data type** – the character's Unicode is cast into the specified numeric type. For example,

  int i = (int)'A'; // i = 65

  Another example:

  int j = '2' + '3';

  In this case, (int)'2" = 50; (int)'3' = 51, therefore, j = 101.

Precedence of Operators

This decides the **order** of evaluation for arithmetic expressions containing several operands.

The list of operators with decreasing priority:

Operator	Meaning	Associativity
`()`	parentheses	left to right
`++, --`	increment, decrement	right to left
`+,-`	unary	right to left
`(Type)`	type cast	right to left
`*, /, %`	multiplication, division, modulus	left to right
`+,-,+`	binary addition, subtraction, String concatenation	left to right
`=,+=,-=,*=,/=`	assignment	right to left

decr

Precedence of Operators

Example:

```
a = ((x + y) * (x – y)) / z;
```

(1) The **parentheses** in (x + y) and (x – y) are evaluated.
(2) Evaluation is done in the **direction** from left to right.
(3) The parentheses of the two results with the **multiplication** operator will be evaluated.
(4) The **division** will be evaluated.

Always safe: Use Parenthesis!!!

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- Constants
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- **Programming Style**
- Case Study

Programming Style

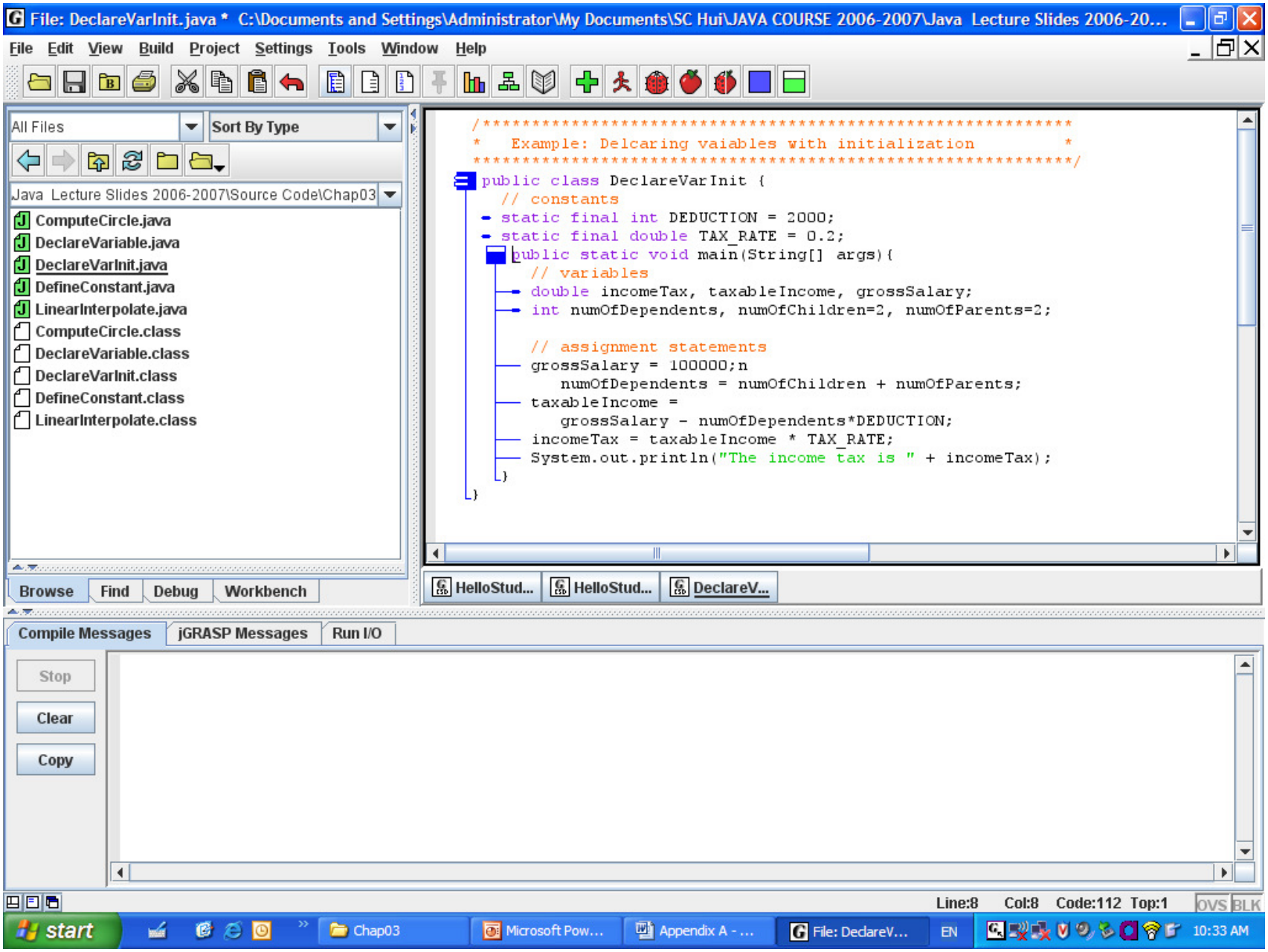
- Make programs simple, readable and easy to understand.
- Using Proper Programming Style
  - should use **indentation**, **blank spaces**, **blank lines**.
  - e.g. using **indentation** as follows:

Use Generate CSD in jGRASP

```
public class DeclareVarInit {
   // constants
   static final int DEDUCTION = 2000;
   static final double TAX_RATE = 0.2;
   public static void main(String[] args){
      // variables
      int numOfChildren=2, numOfParents=2, numOfDependents;
      double grossSalary, taxableIncome, incomeTax;

      // assignment statements
      grossSalary = 100000;
      numOfDependents = numOfChildren + numOfParents;
      taxableIncome =
        grossSalary - numOfDependents*DEDUCTION;
      incomeTax = taxableIncome * TAX_RATE;
      System.out.println("The income tax is " + incomeTax);
   }
}
```

Using Generate CSD in jGRASP



Programming Style

- **Using Meaningful Names**
  1. **Class names** – The first letter of each word in the class name should be capitalized (e.g. **CurrencyExchange**).

  2. **Variable, method and package names** – Lowercase letters should be used for variable and method names. If the name consists of several words, we then use lowercase for the first word, and use capital letter for the first letter of subsequent words, e.g. **convertDollars**().

  3. **Constants** – Uppercase letters should be used for symbolic constants (e.g. **MAXIMUM**), and underscores should be used to separate between words (e.g. **MAX_LIMIT**).

Programming Style

- Adding **comments** into the program.

- At the **beginning of the program**, put comments to say what the program will do; your name and date. Subsequently when the program is modified/extended, put comments to record the modifications, who made it, and the date of the modifications.

- If the purpose of a **portion of the program** is not very clear at a glance, e.g. a nested loop or nested-if statement, put comments to say what the program is doing.

Any special value? Any assumption? Any Pre-condition?

Data and Operators

- Computer Memory
- Data Types
- Literals
- Identifiers
- Constants
- Variables
- Fundamental Arithmetic Operators
- Data Type Conversion
- Programming Style
- **Case Study**

Case Study: Counting Coins

Problem Specification

"Write a program to read a collection of coins and count the total value of the coins into dollars and cents. "

NOTE:

- input and output information are given
- but the formula is not given directly

Problem Analysis


```
amountOfDollars = totalCents / 100;
amountOfCents = totalCents  % 100;
```

Required inputs:

- 50 cents, 20 cents, 10 cents, 5 cents and 1 cent

Required output:

- total value in dollars and cents

FORM VARIABLES ?

Formulas:

- total in cents = ten cents * 10 + twenty cents * 20 + five cents * 5 + fifty cents * 50 + one cent;
- counted dollars = total in cents / 100;
- counted cents = total in cents % 100;

Program Design

Initial Algorithm

LOGIC IN SEQUENCE

1. Read the amount on different types of coins. *HOW??*
2. Compute the total value of coins in cents.
3. Convert the total value into dollars and cents.
4. Print the total value of input coins in dollars and cents.

Program Design

Algorithm in Pseudocode

LOGIC IN SEQUENCE

```
main
   READ fiftyCents,twentyCents,tenCents,
      fiveCents, oneCent

   COMPUTE totalCents = fiftyCents*50 +
      twentyCents*20 + tenCents* 10 +
      fiveCents*5 + oneCents
   COMPUTE countedDollars = totalCents / 100
   COMPUTE countedCents = totalCents % 100

   PRINT countedDollars, countedCents
```

Program Design

Program Dry-run

```
Inputs:
   fiftyCents  = 10, twentyCents = 11,
   tenCents = 12, fiveCents = 13,
   oneCent = 14

totalCents = 10*50 + 11*20 + 12*10 + 13*5 + 14
   = 919
countedDollars = 919 / 100 = 9
countedCents = 919 % 100 = 19

Output:
   The total value of your coins is 9
   dollars and 19 cents.
```

Implementation

```
import java.util.Scanner;
public class CountingCoins
{
    public static void main(String[] args)
    {
      int fiftyCents, twentyCents, tenCents, fiveCents;
      int oneCent, totalCents;
      int countedDollars, countedCents;
      Scanner sc = new Scanner(System.in);
      /* read the amounts on different types of coins */
      System.out.print("Enter number of fifty cents: ");
      fiftyCents = sc.nextInt();
      System.out.print("Enter number of twenty cents: ");
      twentyCents = sc.nextInt();
      System.out.print("Enter number of ten cents: ");
      tenCents = sc.nextInt();
      System.out.print("Enter number of five cents: ");
      fiveCents = sc.nextInt();
      System.out.print("Enter number of one cent: ");
      oneCent = sc.nextInt();
```

VARIABLES

Implementation

```
/* Compute total value of coins into cents */
 totalCents = fiftyCents*50 + twentyCents*20 +
      tenCents*10 + fiveCents*5 + oneCent;

/* Convert total value into dollars and cents */
countedDollars = totalCents / 100;    // division operator
 countedCents = totalCents % 100;   // modulus operator

/* Print the result */
System.out.println("The total value of your coins is " +
countedDollars + " dollars and " + countedCents + " cents" );
  }
}
```

Testing

Program input and output

Enter number of fifty cents: *10*
Enter number of twenty cents: *11*
Enter number of ten cents: *12*
Enter number of five cents: *13*
Enter number of one cent: *14*
The total value of your coins is 9 dollars and 19 cents

Review Exercise: Linear Interpolation

Problem Specification

"The equation of a straight line is given as **y=mx+c**, where *m* is the *slope* and *c* is the *intercept*. If the coordinates of two points P_1 and P_2 are (x_1, y_1) and (x_2, y_2), the slope and the intercept of the line are given in the following equations:

Slope: $$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Intercept: $$c = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

Write a program to ***read in two points***. Then, it finds the ***slope and intercept*** of the straight line given by the coordinates of the two points on the straight line. In addition, the program will also ***read in the x-coordinate*** **of a point**, and then find the ***y-coordinate of the point*** on the straight line. "

Problem Analysis

Required inputs: _____, _____ and _____
Required output: _____ and _____
Formulas: slope = _____,
intercept = _____,
y = _____

System Design

Initial Algorithm: (Exercise)
Algorithm in Pseudocode

main()

ANY VARIABLES ??

READ __, __, __, ___, ____
COMPUTE _____ = _____
COMPUTE _____ = _____
COMPUTE _____ = _____
PRINT _____. _____. _____

Implementation

```
Import java.util.Scanner;
public class LinearInterpolate {
    public static void main(String[] args){
        // Declare variables

        double __, __, __, __, __, __, __, __;
        Scanner sc = new Scanner(System.in);

        // Read the two points & x-coordinate

        System.out.print("Enter first point x1 y1: ");
        __ = sc.nextDouble();
        __ = sc.nextDouble();

        System.out.print("Enter second point x2 y2: ");
        __ = sc.nextDouble();
        __ = sc.nextDouble();

        System.out.print("Enter x-coordinate of pt: ");
        __ = sc.nextDouble();
```

VARIABLES

Implementation

```
        // Calculate the slope, intercept, y-ccordinate

        __ = _______;
        __ = _______;
        __ = _______;

        // Print slope, intercept, y-ccordinate

        System.out.println("The ___ is " + ___);
        System.out.println("The ___ is " + ___);
        System.out.println("The ___ is " + ___);
    }
}
```

Exercise Solution

Key Terms

- operator
- arithmetic operator
- assignment operator
- decremental operator (--)
- incremental operator (++)
- concatenation operator
- operator precedence
- expression
- data type conversion
- package, import
- Math class
- Wrapper class
- Autoboxing, unboxing
- Enumerated type
- DecimalFormat class

Further Reading

- Read Chapter 3 on “Data Types, Constants and Variables” of the textbook
- Read Chapter 5 on “Operators, Expressions and Assignments” of the textbook
- Read Section 5.13 of the textbook on other case studies.