JAVA PROGRAMMING

Week 1: Data Types and Operators

Lecturers:

- Hồ Tuấn Thanh, M.Sc.

Why data types are important ?

- Data types are especially important in Java because it is a strongly typed language.
  - This means that all operations are type-checked by the compiler for type compatibility.
- Strong type checking helps prevent errors and enhances reliability.
  - To enable strong type checking, all variables, expressions, and values have a type.
- The type of a value determines what operations are allowed on it.
  - An operation allowed on one type might not be allowed on another.

Java's primitive types

- Two general categories of built-in data types:
  - object-oriented and
  - non object-oriented.

Java's Built-in Primitive Data Types

Type	Meaning
boolean	Represents true/false values
byte	8-bit integer
char	Character
double	Double-precision floating point
float	Single-precision floating point
int	Integer
long	Long integer
short	Short integer

Integers

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

Example


```
/* Compute the number of cubic inches
 * in 1 cubic mile. 1 mile = 63360 inches
 */
public class Inches {
        public static void main(String[] args) {
                long ci;
                long im;
                im = 63360;
                ci = im * im * im;
                System.out.println("There are " + ci
                                        + " cubic inches in cubic mile.");
        }
}
```

```
There are 254358061056000 cubic inches in cubic mile.
```

Floating-Point Types

- The floating-point types can represent numbers that have fractional components.
- Two floating-point types:
  - float (32 bits) and double (64 bits).
- double is the most commonly used, and many of the math functions in Java's class library use double values.

Example


```
/* Use the Pythagorean theorem to
 * find the length of the hypotenuse
 * given the lengths of the two opposing sides.
 */
public class Hypot {

    public static void main(String[] args) {
        double x, y, z;
        x = 3;
        y = 4;
        z = Math.sqrt(x*x + y*y);
        System.out.println("Hypotenuse is " + z);
    }
}
```

```
Hypotenuse is 5.0
```

Characters

- Java uses Unicode.
- Unicode defines a character set that can represent all of the characters found in all human languages.
- char is an unsigned 16-bit type having a range of 0 to 65,535.
- The standard 8-bit ASCII character set is a subset of Unicode and ranges from 0 to 127.
  → The ASCII characters are still valid Java characters.
- Example:

```
char ch;
ch = 'X';
System.out.println("This is ch:" + ch);
```

Why does Java use Unicode?

Example


```
// Character variables can be handled like integers.
public class CharArithDemo {
    public static void main(String[] args) {
        char ch;
        ch = 'X';
        System.out.println("ch contains " + ch);
        ch++; // increment ch
        System.out.println("ch is now " + ch);
        ch = 90; // give ch the value Z
        System.out.println("ch is now " + ch);
    }
}
```

```
ch contains X
ch is now Y
ch is now Z
```

Boolean type

- The Boolean type represents true/false values.
- Example:

```
// Demonstrate boolean values.
public class BoolDemo {
	public static void main(String[] args) {
		boolean b;
		b = false;
		System.out.println("b is " + b);
		b = true;
		System.out.println("b is " + b);
		// a boolean value can control the if statement
		if(b) System.out.println("This is executed.");
		b = false;
		if(b) System.out.println("This is not executed.");
		// outcome of a relational operator is a boolean value
		System.out.println("10 > 9 is " + (10 > 9));
	}
}
```

```
b is false
b is true
This is executed.
10 > 9 is true
```

Exercise: How Far Away Is the Lightning?

1. Create a new file called Sound.java.
2. Write a program that computes how far away, in **feet**, a listener is from a lightning strike. Sound travels approximately 1,100 feet per second through air. Thus, knowing the interval between the time you see a lightning bolt and the time the sound reaches you enables you to compute the distance to the lightning. We assume that the time interval is 7.2 seconds.

```
/* Compute the distance to a lightning
 * strike whose sound takes 7.2 seconds to reach you
 */
public class Sound {
	public static void main(String[] args) {
		double dist;
		dist = 7.2 * 1100;
		System.out.println("The lightning is " + dist +
		                                   " feet away.");
	}
}
```

```
The lightning is 7920.0 feet away.
```

Literals

- Literals refer to fixed values that are represented in their human-readable form.
- Literals are also commonly called constants.
- Java literals can be of any of the primitive data types. The way each literal is represented depends upon its type.
- Example:
  - 10 and -100 are integer literals.
  - 11.123 is a floating point literal.
  - 12 is an int, but 12L is a long.
  - By default, floating-point literals are of type double. To specify a float literal, append an F or f to the constant (10.19F).

Hexadecimal, Octal, and Binary Literals

- A hexadecimal literal must begin with `0x` or `0X`
- An octal literal begins with a zero.
- A binary literal begins with a `0b` or `0B`.

- Example:
  - `int hex = 0xFF; // 255 in decimal`
  - `int oct = 011; // 9 in decimal`
  - `int binary = 0b1100; // 12 in decimal`

Character Escape Sequences

Escape Sequence	Description
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line
\f	Form feed
\t	Horizontal tab
\b	Backspace
ddd	Octal constant (where *ddd* is an octal constant)
\u*xxxx*	Hexadecimal constant (where *xxxx* is a hexadecimal constant)

String Literals

- A string is a set of characters enclosed by double quotes.
- A string literal can also contain one or more of the escape sequences .
- Example:

```
// Demonstrate escape sequences in strings
public class StrDemo {
        public static void main(String[] args) {
                System.out.println("First line\nSecond line");
                System.out.println("A\tB\tC");
                System.out.println("D\tE\tF");
        }
}
```

```
First line
Second line
A       B       C
D       E       F
```

Variables

type varname;

- where type is the data type of the variable, and varname is its name
- Initializing a Variable:

type var = value;

- Example:

```
int count = 10; // give count an initial value of 10
char ch = 'X'; // initialize ch with the letter X
float f = 1.2F; // f is initialized with 1.2
int a, b = 8, c = 19, d; // b and c have initializations
```

Dynamic initialization


```
// Demonstrate dynamic initialization.
// Compute volume of the cylinder
public class DynInit {

	public static void main(String[] args) {
		double radius = 4, height = 5;
		//dynamically initialize volume
		double volume = 3.1416 * radius * radius * height;
		System.out.println("Volume is " + volume);
	}
}
```

Scope and lifetime of variables

- Java allows variables to be declared within any block.
- A block (is begun with an opening curly brace and ended by a closing curly brace) defines a scope.
  - A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.
- In general, every declaration in Java has a scope.
- Two of the most common scopes in Java:
  - defined by a class and
  - defined by a method.
- General rule: variables declared inside a scope are not visible (that is, accessible) to code that is defined outside that scope.

Example: Scope


```
// Demonstrate block scope
public class ScopeDemo {
    public static void main(String[] args) {
        int x; // known to all code within main
        x = 10;
        if(x == 10) { // start new scope
            int y = 20; // know only to this block
            // x and y both known here.
            System.out.println("x and y: " + x + " " + y);
            x = y * 2;
        }
        // y == 100; // Error! y is not known here
        // x is still known here.
        System.out.println("x is " + x);
    }
}
```

Example: Lifetime


```
// Demonstrate lifetime of a variable.
public class VarInitDemo {
	public static void main(String[] args) {
		int x;
		for(x = 0; x < 3; x++) {
			int y = -1;
			// y is initialized each time block is entered
			System.out.println("y is : " + y);
			// this always prints -1
			y = 100;
			System.out.println("y is now: "+ y);
		}
	}
}
```

Operators

- An operator is a symbol that tells the compiler to perform a specific mathematical or logical manipulation.
- Java has four general classes of operators:
  - arithmetic,
  - bitwise, relational,
  - and logical.

Arithmetic operators

Operator	Meaning
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
– –	Decrement

Example


```
// Demonstrate the % operator
public class ModDemo {
    public static void main(String[] args) {
        int iresult, irem;
        double dresult, drem;
        iresult = 10 / 3;
        irem = 10 % 3;
        dresult = 10.0 / 3.0;
        drem = 10.0 % 3.0;
        System.out.println("Result and remainder of 10 / 3: " +
                                    iresult + " " + irem);
        System.out.println("Result and remainder of 10.0/3.0: "+
                                    dresult + " " + drem);
    }
}
```

```
Result and remainder of 10 / 3: 3 1
Result and remainder of 10.0 / 3.0: 3.3333333333333335 1.0
```

Increment and Decrement

- x++; is the same as x = x + 1;
- x--; is the same as x = x - 1;
- Both the increment and decrement operators can either precede (prefix) or follow (postfix) the operand.
- For example: x = x + 1; can be written as

```
++x; // prefix form
```

or

```
x++; // postfix form
```

- Consider the following:

```
x = 10;                    and          x = 10;
y = x++;                                y = ++x;
```

Relational and logical operators

Operator	Meaning
= =	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Operator	Meaning
&	AND
|	OR
^	XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	NOT

p	q	p & q	p | q	p ^ q	!p
False	False	False	False	False	True
True	False	False	True	True	False
False	True	False	True	True	True
True	True	True	True	False	False

```
// Demonstrate the relational and logical operators.
public class RelLogOps {
    public static void main(String[] args) {
        int i, j;
        boolean b1, b2;
        i = 10;
        j = 11;
        if(i < j) System.out.println("i < j");
        if(i <= j) System.out.println("i <= j");
        if(i != j) System.out.println("i != j");
        if(i == j) System.out.println("This won't execute");
        if(i >= j) System.out.println("This won't execute");
        if(i > j) System.out.println("This won't execute");
        b1 = true;
        b2 = false;
        if(b1 & b2) System.out.println("This won't execute");
        if(!(b1 & b2)) System.out.println("!(b1 & b2) is true");
        if(b1 | b2) System.out.println("b1 | b2 is true");
        if(b1 ^ b2) System.out.println("b1 ^ b2 is true");
    }
}
```

```
i < j
i <= j
i != j
!(b1 & b2) is true
b1 | b2 is true
b1 ^ b2 is true
```

Short-circuit logical operators

- Java supplies special short-circuit versions of its AND and OR logical operators that can be used to produce more efficient code.
  - In an AND operation, if the first operand is false, the outcome is false no matter what value the second operand has.
  - In an OR operation, if the first operand is true, the outcome of the operation is true no matter what the value of the second operand.
  - Thus, in these two cases there is no need to evaluate the second operand. → time is saved and more efficient code is produced.
- The short-circuit AND (OR) operator is && (||). Their normal counterparts are & and |.

Example


```
// Demonstrate the short-circuit operators.
public class SCops {
	public static void main(String[] args) {
		int n, d, q;
		n = 10;
		d = 2;
		if(d != 0 && (n %d ) == 0)
			System.out.println(d + " is a factor of " + n);
		d = 0; // now, set d to zero
		// Since d is zero, the second operand is not evaluated
		if(d != 0 && (n % d) == 0)
			System.out.println(d + " is a factor of " + n);
		/* Now, try same thing without short-circuit operator.
		 * This will cause a divide-by-zero error.
		 */
		if(d != 0 & (n % d) == 0)
			System.out.println(d + " is a factor of " + n);
	}
}
```

Assignment operator

var = expression;

- The type of var must be compatible with the type of expression.
- Example:

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

Shorthand assignments

- Java provides special shorthand assignment operators that simplify the coding of certain assignment statements.

x = x + 10;

can be written, using Java shorthand, as

x += 10;

+=	-=	*=	/=
%=	&=	|=	^=

Example: side effect


```
// Side effects can be important.
public class SideEffects {
    public static void main(String[] args) {
        int i;
        i = 0;
        /* Here, i is still incremented even though
        * the if statement fails. */
        if(false & (++i < 100))
        System.out.println("This won't be displayed");
        System.out.println("if statement executed: " + i); //displays 1
        /* In this case, i is not incremented because
        * the short-circuit operator skips the increment. */
        if(false && (++i < 100))
        System.out.println("This won't be displayed");
        System.out.println("if statement executed: " + i); //still 1!
    }
}
```

Type conversion in assignments

- When one type of data is assigned to another type of variable, an automatic type conversion will take place if
  - The two types are compatible.
  - The destination type is larger than the source type.

Example


```
// Demonstrate automatic conversion from long to double
public class LtoD {
	public static void main(String[] args) {
		long L;
		double D;
		L = 100123285L;
		D = L; // automatic conversion from long to double
		System.out.println("L and D: " + L + " " + D);
	}
}
```

```
L and D: 100123285 1.00123285E8
```

```
// This program will not compile.
public class DtoL {

    public static void main(String[] args) {
        long L;
        double D;
        D = 100123258.0;
        L = D; // Illegal! No automatic conversion
                    // from double to long
        System.out.println("L and D: " + L + " " + D);
    }
}
```

Casting incompatible types

(target-type) expression

- target-type specifies the desired type to convert the specified expression to.
- Example:

```
double x = 3.0, y = 2.0;
// .....
int z = (int)(x / y);
```

- When a cast involves a narrowing conversion, information might be lost.
  - Example: when casting a long into a short, information will be lost if the long's value is greater than the range of a short because its high-order bits are removed.

```java
// Demonstrate casting.
public class CastDemo {
	public static void main(String[] args) {
		double x, y;
		byte b;
		int i;
		char ch;
		x = 10.0;
		y = 3.0;
		i = (int)( x / y); //cast double to int
		System.out.println("Integer outcome of x / y: " + i);
		i = 100;
		b = (byte) i;
		System.out.println("Value of b: " + b);
		i = 257;
		b = (byte) i;
		System.out.println("Value of b: " + b);
		b = 88; // ASCII code for X
		ch = (char) b;
		System.out.println("Value of ch: " + ch);
	}
}
```

```
Integer outcome of x / y: 3
Value of b: 100
Value of b: 1
Value of ch: X
```

Operator precedence

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Exercise: Display a Truth Table for the Logical Operators

1. Create a new file called LogicalOpTable.java.
2. To ensure that the columns line up, you will use the \t escape sequence to embed tabs into each output string.
3. Each subsequent line in the table will use tabs to position the outcome of each operation under its proper heading.
4. Put your implementation in the file LogicalOpTable.java.
5. Compile and run the program.
6. Try modifying the program so that it uses and displays 1's and 0's, rather than true and false.

Result


```
P       Q       AND     OR      XOR     NOT
1       1       1       1       0       0
1       0       0       1       1       0
0       1       0       1       1       1
0       0       0       0       0       1


P       Q       AND     OR      XOR     NOT
true    true    true    true    false   false
true    false   false   true    true    false
false   true    false   true    true    true
false   false   false   false   false   true
```

Type Conversion in Expressions

- Within an expression, it is possible to mix two or more different types of data as long as they are compatible with each other.
- Example: you can mix short and long within an expression because they are both numeric types.
- When different types of data are mixed within an expression, they are all converted to the same type
  - Example: First, all char, byte, and short values are promoted to int. Then, if one operand is a long, the whole expression is promoted to long. If one operand is a float operand, the entire expression is promoted to float. If any of the operands is double, the result is double.

Example: Promotion


```
// Demonstrate a promotion
public class PromDemo {
        public static void main(String[] args) {
                byte b;
                int i;
                b = 10;
                i = b * b; // Ok, no cast needed
                b = 10;
                // Cast is needed here to assign an int to a byte
                b = (byte) (b * b);
                System.out.println("i and b: " + i + " " + b);
        }
}
```

Example: Using a cast


```
// Using a cast
public class UseCast {
      public static void main(String[] args) {
            int i;
            for(i = 0; i < 5; i++) {
            System.out.println(i + " / 3: " + i / 3);
            System.out.println(i + " / 3 with fractions: "
                                                + (double)i / 3);
            }
      }
}
```

```
0 / 3: 0
0 / 3 with fractions: 0.0
1 / 3: 0
1 / 3 with fractions: 0.3333333333333333
2 / 3: 0
2 / 3 with fractions: 0.6666666666666666
3 / 3: 1
3 / 3 with fractions: 1.0
4 / 3: 1
4 / 3 with fractions: 1.3333333333333333
```

Spacing and Parentheses

- An expression in Java may have tabs and spaces in it to make it more readable.

```
x=10/y*(127/x);
x = 10 / y * (127/x);
```

- Parentheses increase the precedence of the operations contained within them.
- Use of redundant or additional parentheses will not cause errors or slow down the execution of the expression.

→You are encouraged to use parentheses to make clear the exact order of evaluation.

```
x=y/3-34*temp+127;
x = y/3 - (34*temp) + 127;
```

Exercises

QUESTION ?