

Object Oriented Programming with Java

Zheng Chen



Java Basics

1. Java Program Structure
2. Java Character Set
3. Java Types
4. Literal and Variable
5. Operator and Expression



**KEEP
CALM
AND
CODE
JAVA**

Java Program Structure

- **Package Statement** — It includes statement that provides a package declaration.
- **Import statements** — It includes statements used for referring classes and interfaces that are declared in other packages.
- **Class/Interface Section** — It describes information about user defines classes present in the program. Every Java program consists of at least one class definition. This class definition declares the main method. It is from where the execution of program actually starts.

Java Program Structure — Package Statement

- **package package.name;**
- Java allows you to group classes in a collection known as package.
- It must appear as the first statement in the source code file before any class or interface declaration. This statement is optional.
- Only one package statement per source file.

Java Program Structure — Import Statement

- **import packageName.ClassName;**
- **import packageName.*;**
- It includes statements used for referring classes and interfaces that are declared in **other packages**.
- More than one import statement is legal.
- java.lang package is auto-imported.

Java Program Structure — Class/Interface Section

```
class HelloJava {  
    class body;  
}
```

- Classes are the main and essential elements of any Java program.
- A Java program may contain several class definitions.
- Interfaces are like a class that includes a group of method declarations.

Java Program Structure — Main Method

```
public static void main(String args[]){  
    method body;  
}
```

- Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program.

Java Program Structure — Comment

- Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily.
- **Single line (or end-of line) comment:** It starts with a double slash symbol (//) and terminates at the end of the current line.
- **Multiline Comment:** Java programmer can use C/C++ comment style that begins with delimiter /* and ends with */.
- **Documentation comments:** This comment style is new in Java. Such comments begin with delimiter /** and end with */. The main purpose of this type of comment is to automatically generate program documentation.

Java Basics

1. Java Program Structure
- 2. Java Character Set**
3. Java Types
4. Literal and Variable
5. Operator and Expression



**KEEP
CALM
AND
CODE
JAVA**

Java Charset — Unicode

- A character set is a set of textual and graphic symbols, each of which is mapped to a set of nonnegative integers.
- Java uses the **Unicode** character set for representing character data.
- The Unicode set represents each character as a **16-bit** unsigned integer. It can, therefore, represent $2^{16} = 65,536$ different characters.



Adopt a Character

Emoji

Basic Info


Latest

Connect

Membership

Press

Search ...

3 U+00B3	☐ U+2751	୪ U+0C20	و U+06C4	ঐ U+0B37	満 U+1F235	॥ U+00B6	୩ U+0E95	< U+304F	ğ U+01E7
… U+2026	ष U+0937	 U+3023	~ U+058A	■ U+2B1C	₣ U+056F	ह U+0939	୨ U+13EA	^ U+3078	^ U+02C6
ଐ U+0B14	☐ U+FF9B	Everyone in the world should be able to use their own language on phones and computers. LEARN MORE ABOUT UNICODE						 ADOPT A CHARACTER	
Ⅱ U+1111	👉 U+1F91E	୧ U+10DA	💛 U+1F49B	÷ U+00F7	₹ U+0F4F	😊 U+263B	୩ U+A755	👉 U+261C	✖ U+2716
𐌆 U+178A	✓ U+2714	✍ U+270D	☎ U+2706	👤 U+2603	# U+266F	⌚ U+0298	₹ U+0568	₳ U+10F0	— U+2013
ᱚ U+1570	郷 U+9109	ᱱ U+2690	、 U+3001	👱 U+1F475	♡ U+2661	𑖀 U+26A2	घ U+0918	و U+FEED	ق U+0642
尸 U+2640	𑖀 U+2640	· U+2022	↓ U+2193	୪ U+0C20	🐱 U+1F406	୨ U+0E95	୩ U+0E95	୩ U+0E95	୩ U+0E95

Character Set vs. Character Encoding

- Charset is the set of characters you can use.
- Encoding is the way these characters are stored.

Charactor	ASCII	Unicode	UTF-8	UTF-16	UTF-32
A	41	\u0041	\x41	\u0041	00000041
陈	-	\u9648	\xe9\x99\x88	\u9648	00009648

Java Charset — Tokens

- Java Tokens are the smallest individual building block or smallest unit of a Java program; the Java compiler uses it for constructing expressions and statements. Java program is a collection of different types of **tokens**, **comments**, and **white spaces**.
- There are various tokens used in Java:
 - **Reserved Keywords**
 - **Identifiers**
 - **Literals**
 - **Operators**
 - **Separators**

Tokens — Reserved Keywords

- Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name.

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

* not used

Tokens — Reserved Keywords

- **const** and **goto** are reserved words, even though they are not currently used.
- **true**, **false**, and **null** might seem like keywords, but they are actually literals.
- You cannot use keywords or reserved words as identifiers in your programs.

Tokens — Identifiers

- Identifiers are the names of variables, methods, classes, packages and interfaces.
- All identifiers must start with either a **letter** or currency character(\$) or an underscore(_).
- They must **not** begin with a digit.
- After the first character, an identifier can have any combination of letter, digit, currency character(\$) and an underscore(_).
- A Java keywords cannot be used as an identifier.
- Identifiers in Java are **case sensitive**, foo and Foo are two different identifiers.
- They can be any length.

Identifiers — Legal Identifiers

MyVariable	\$myvariable
myvariable	_9pins
MYVARIABLE	andros
x	\$\$\$\$\$
i	电子科技大学
_myvariable	ヤo電子科技大罌ooO

Identifiers — Illegal Identifiers

My Variable // Contains a space

9pins // Begins with a digit

a+c // The plus sign is not an letter

class // class is a keyword

testing1-2-3 // The hyphen is not an letter

O'Reilly // Apostrophe is not an letter

OReilly_&_Associates // ampersand is not an letter

Identifiers — Naming Conventions

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- CamelCase in java naming conventions

nameNameName....

- But, it is not forced to follow.

Identifiers — Naming Conventions

- **Class names** should be nouns, in mixed case with the first letter of each internal word capitalized. Interfaces name should also be capitalized just like class names.
- Use whole words and must avoid acronyms and abbreviations.

class MountainBike

Identifiers — Naming Conventions

- **Methods** should be verbs, in mixed case with the first letter lowercase and with the first letter of each internal word capitalized.

void changeGear(int newValue);

- **Constant variables** should be all uppercase with words separated by underscores (“_”).

static final int MIN_WIDTH = 4;

Identifiers — Naming Conventions

- **Variable** names should be short yet meaningful.
- Should not start with underscore('_') or dollar sign '\$' characters.
- One-character variable names should be avoided except for temporary variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

int speed = 0;

Identifiers — Naming Conventions

- **package name** is always written in all-lowercase ASCII letters and should be one of the top-level domain names, like com, edu, gov, mil, net, org.
- Subsequent components of the package name vary according to an organization's own internal naming conventions.

edu.uestc.java

Java Basics

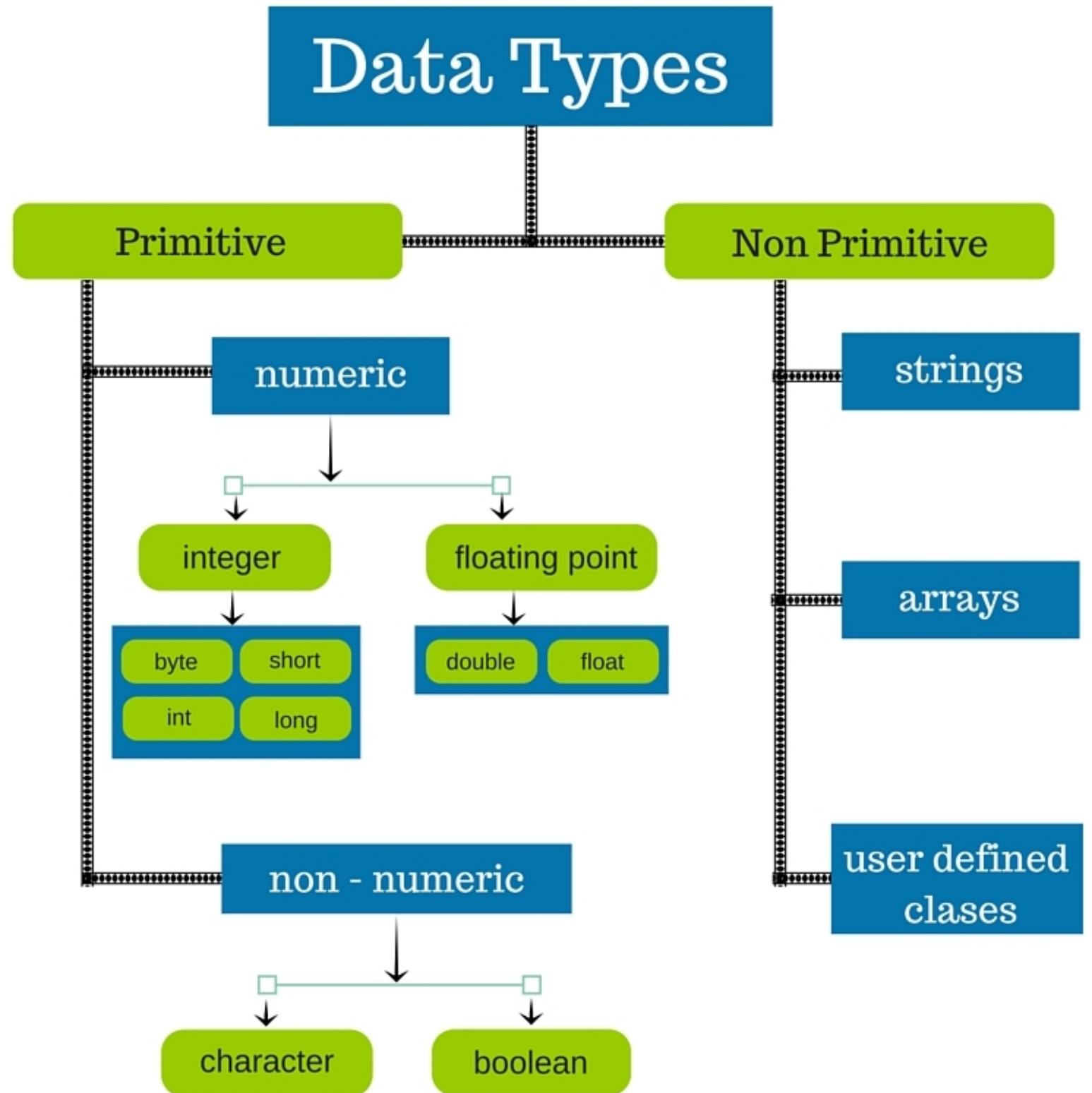
1. Java Program Structure
2. Java Character Set
- 3. Java Data Types**
4. Literal and Variable
5. Operator and Expression



KEEP
CALM
AND
CODE
JAVA

Java Data Types

- Data types specify the different sizes and values that can be stored in the variable.



Java Data Types — Primitive

Data Type	Default Value	Default size
boolean	FALSE	1 bit
char	\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Java Data Types — Primitive

- An **integer** primitive is one whose value can be only an integer. It cannot be a real number (that is, one with a fractional value).
- The distinguishing feature between the various integer primitives is how many bits make up each one.
- Integer primitives are signed.

Type	Size	Range
byte	8 bits	-128 .. 127
short	16 bits	-32,768 .. 32,767
int	32 bits	-2,147,483,648 .. 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

Java Data Types — Primitive

- Java supports two real primitives, float and double. **float** gets its name from the idea of a floating point number. The decimal point can move, so it's said to “float.” **double** gets its name because it takes twice the storage space of a float.

Type	Size	Range
float	32 bits	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$
double	64 bits	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$

Java Data Types — Primitive

- The **boolean** data type indicates whether something is true or false. In fact, those two words (true and false) are the only two values boolean data types can have.
- The **char** data type holds two 8-bit bytes and is meant to represent characters. It's stored as an unsigned 16-bit integer with a minimum value of 0 and a maximum value of 65,535. However, you should never use a char to store a number.

Java Data Types — Non Primitive

- **Derived data types** are those whose variables allow us to store multiple values of same type. But they never allows to store multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieve using array.
- There're two derived data types: **Array** and **String**.

Java Data Types — Non Primitive

- **Reference data types** are those which are developed by programmers by making use of appropriate features of the language. Reference variables are created using defined constructors of the classes. They are used to access objects.

Car myCar=new Car("Lamborghini");

Java Basics

1. Java Program Structure
2. Java Character Set
3. Java Data Types
4. **Literal and Variable**
5. Operator and Expression



**KEEP
CALM
AND
CODE
JAVA**

Literal and Variable — Literal

- A literal is a constant value that corresponds to a particular data type.

`int a = 100; // Here 100 is a literal`

`int b = 0100; // octal-form literal`

`int c = 0xFace; // Hexa-decimal form literal`

`int d = 0b1111; // Binary literal`

Literal and Variable — Literal

- Literals use appending character to determine data types when necessary.

long a = 100L; // L indicates a long literal

float b = 0.1F; // F indicates a float literal

double c = 0.2D; // D indicates a double literal

Literal and Variable — Literal

- For char data types we can specify literals in 4 ways:

- **Single quote** : `char ch = 'a';`

- Char literal **as Integral** literal : `char ch = 062;`

- **Unicode Representation** :

`char ch = '\u0061';` // Here `\u0061` represent a.

- **Escape Sequence** : Every escape character can be specify as char literals.

`char ch = '\n';`

Literal — Escape Sequence

Escape Sequence	Effect
\'	Create a single quotation mark
\"	Create a double quotation mark
\\	Create a backslash character
\n	Create a new line (often called the newline character)
\t	Create a tab
\b	Create a backspace character
\r	Return to the start of the line (but do not make a new line)
\f	Form feed (move to the top of the next page for printers)
\a	The alert (or bell) character

Literal and Variable — Literal

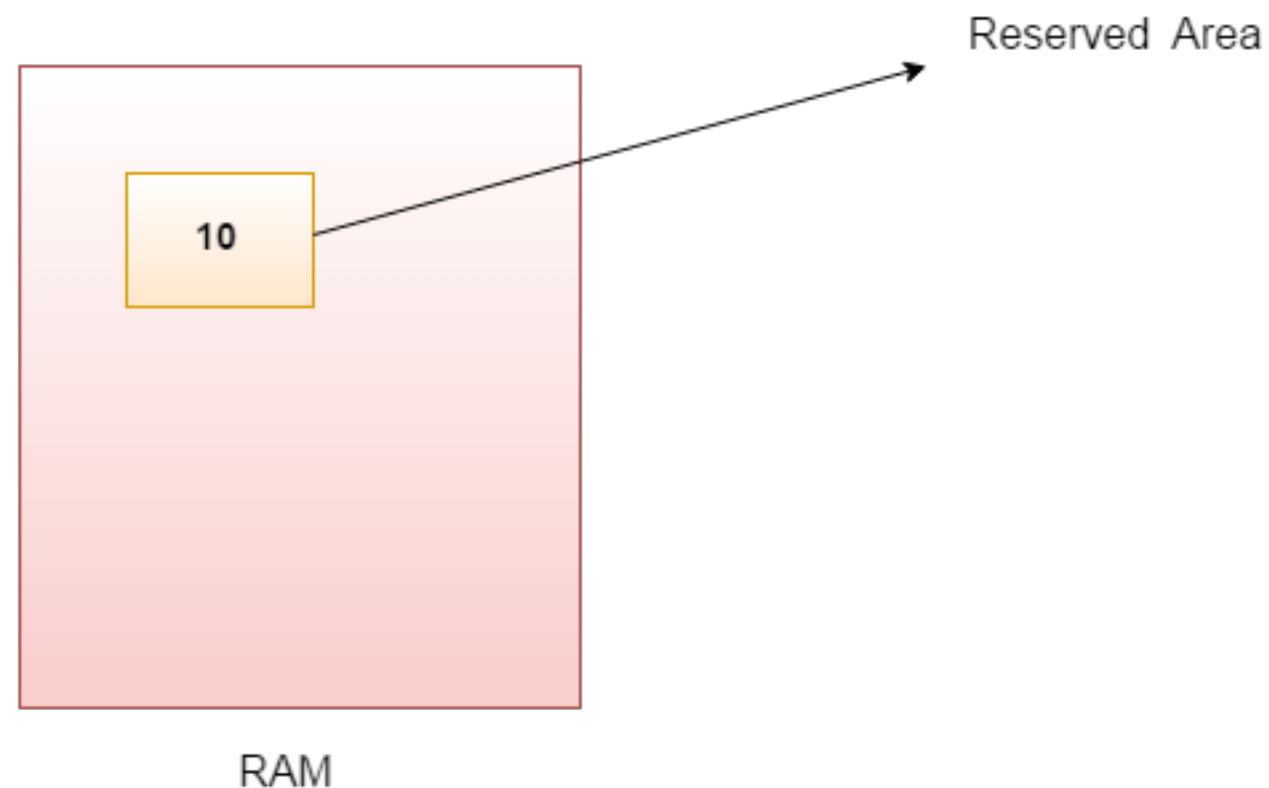
- A **String literal** is a sequence of characters used by Java programmers to populate String objects or display text to a user. The characters could be letters, numbers or symbols and are enclosed within two quotation marks.

"So my friend said, \"It's how big?\""

- There are two **boolean literals**: **true** represents a true boolean value, **false** represents a false boolean value.

Literal and Variable — Variable

- **Variable** is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.



- `int data=50; //Here data is variable`

Literal and Variable — Variable

- Java is considered as a **strongly typed** programming language. To create a variable, you must specify the type, initialValue is optional:

type identifier [= value] [,identifier [= value]];

- Here are some examples:

```
int myNum;
```

```
float myFloatNum = 5.99f;
```

```
char myLetter = 'D';
```

```
boolean myBool = true;
```

```
String myText = "Hello";
```

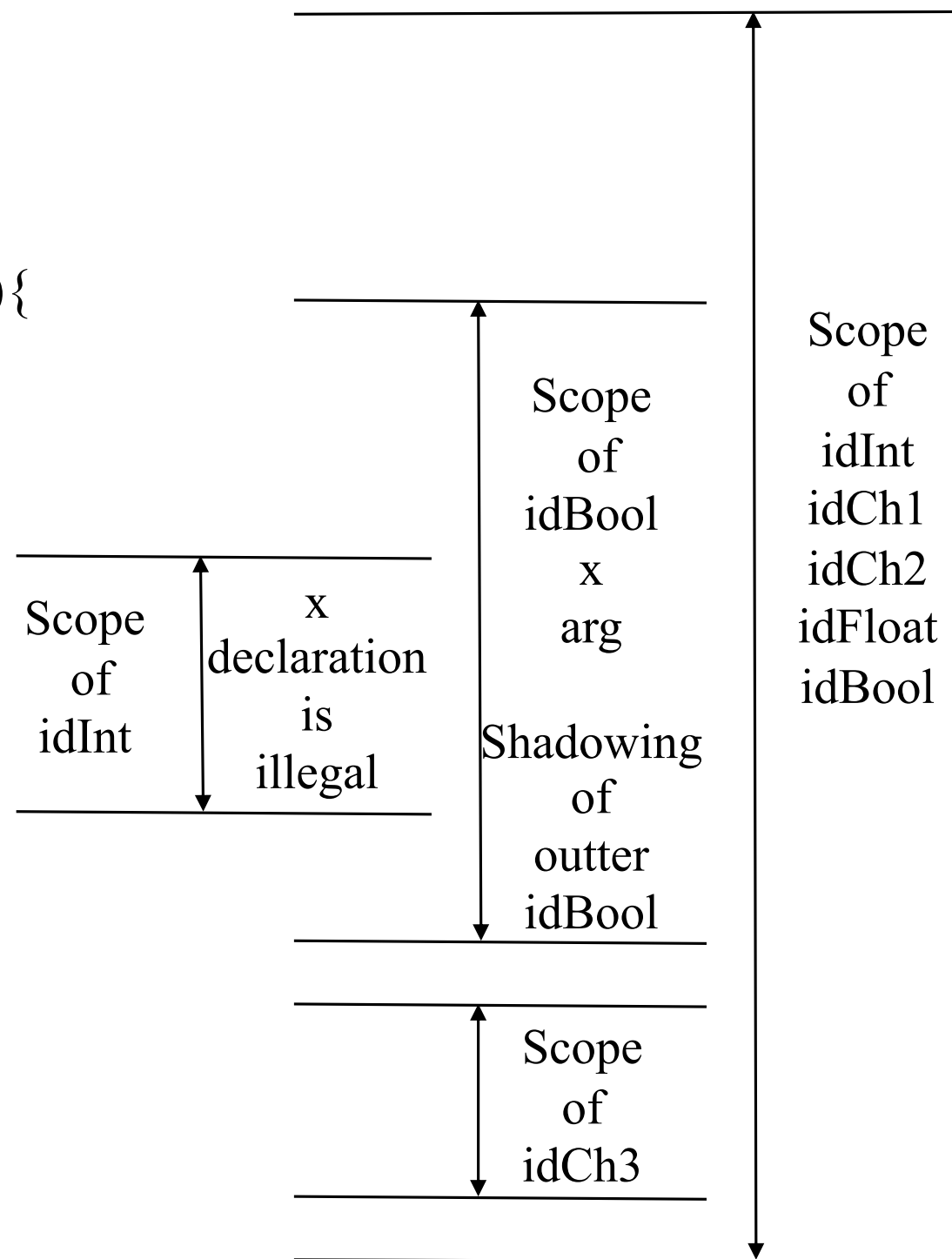
Variable — Scope of Variable

- Scope of a variable is the part of the program where the variable is accessible.
- **Class Level Scope:** variables declared inside class (outside any function) can be directly accessed anywhere in class.
- **Method Level Scope:** variables declared inside a method have method level scope and can't be accessed outside the method.
- **Block Scope:** variables declared inside pair of brackets “{” and “}” in a method have scope within the brackets only.


```

class AppExa{
    int idInt;
    char idCh1,idCh2;
    float idFloat;
    boolean idBool ;
    public static void main(String arg[]){
        boolean idBool = true;
        int x = 10;
        . . .
        while (idBool) {
            int x = 11;
            int idInt=1;
            . . .
        }
        . . .
    }
    public void testProg(){
        char idCh3;
        . . .
    }
}

```



Variable — Variable Initialization

- In Java, the compiler checks to make sure that you have **assigned** a value **before** you use a variable.
- But, class/instance variables can be auto initialized.

Data Type	Default Value
boolean	FALSE
char	\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d

Java Basics

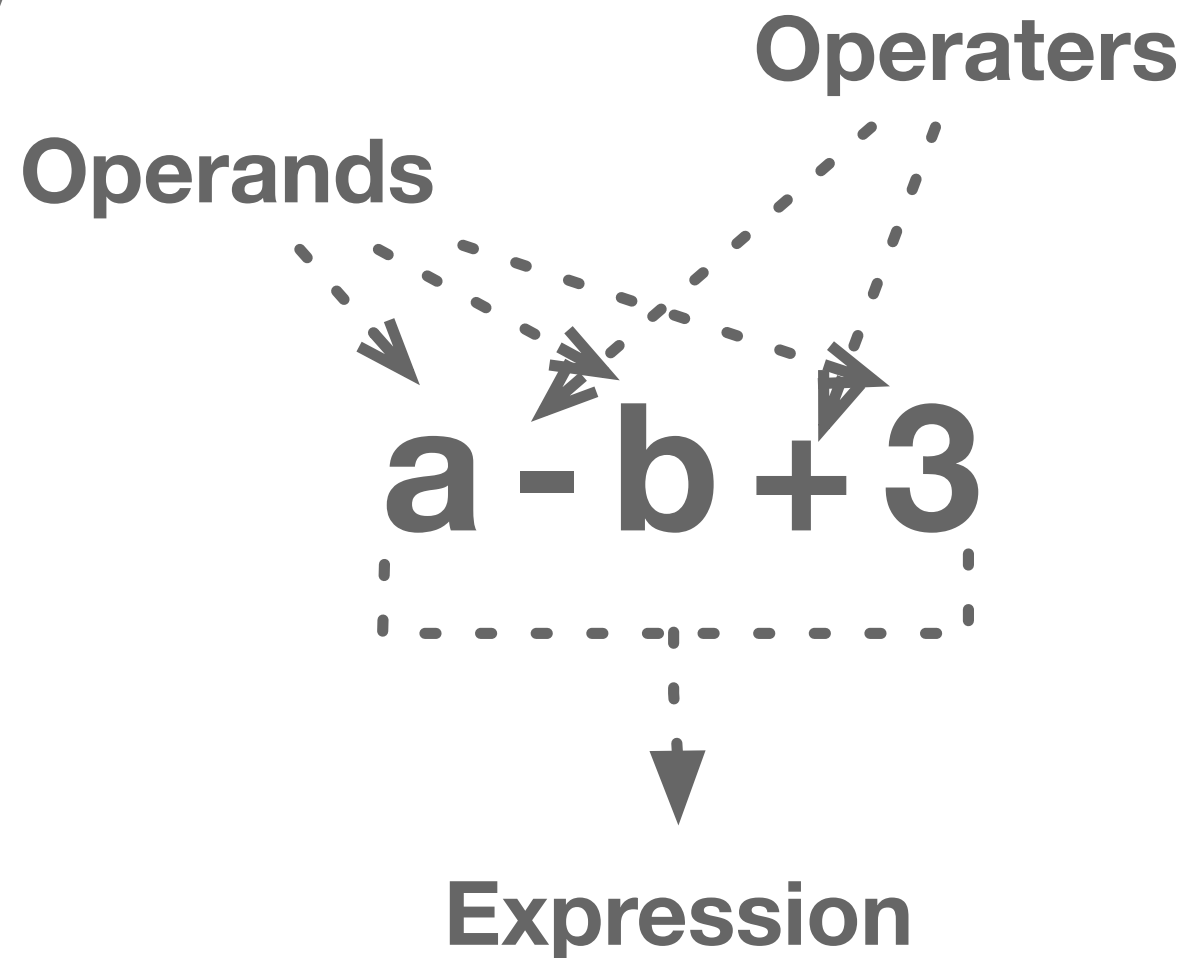
1. Java Program Structure
2. Java Character Set
3. Java Data Types
4. Literal and Variable
5. **Operator and Expression**



KEEP
CALM
AND
CODE
JAVA

Operator and Expression

- An **expression** is a statement that can convey a value.
- **Operators** are special symbols (characters) that carry out operations on operands (variables and values).



Operator and Expression — Operators

- **Arithmetic operators** are used in mathematical expressions in the same way that they are used in algebra.
- **Relational Operators** determines the relationship between two operands.
- **Bitwise Operators** perform bitwise or bit shift operations on integral types.
- **Logical operators** are used in decision making and looping.
- **Assignment Operators** are used to assign values to variables.
- **Miscellaneous Operators** are few other operators.

Operators — Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Assume integer variable A holds 10 and variable B holds 20

Operators — Relational Operators

Operator	Description	Example
==(equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Assume integer variable A holds 10 and variable B holds 20

Operators — Bitwise Operators

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
(bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Assume integer variable A holds 60 and variable B holds 13

Operators — Logical Operators

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
(logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Assume Boolean variables A holds true and variable B holds false

Operators — Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	$C \% = A$ is equivalent to $C = C \% A$

Operators — Assignment Operators 2

Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise and AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
=	bitwise inclusive OR and assignment operator.	C = 2 is same as C = C 2

Miscellaneous Operators

- **Conditional Operator (? :)** is shorthand for if-then-else statement.

variable x = (expression) ? value if true : value if false

- **instanceof Operator** checks whether the object is of a particular type (class type or interface type).

(Object reference variable) instanceof (class/interface type)

Operators — Precedence of Java Operators

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.

Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression --expression +expression -	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	= !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	+= -= *= /= %= ^= = <<= >>= >>>=	Right to left

Homework

- Produce a formatted 12×12 multiplication table

[illegible]