



# LECTURE 02

# Data Types, Decision & Iteration

**ITCS123 Object Oriented Programming**

Dr. Siripen Pongpaichet

Dr. Petch Sajjacholapunt

Asst. Prof. Dr. Ananta Srisuphab

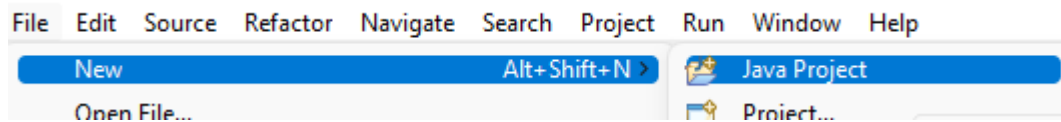


# Recap – Lecture 01

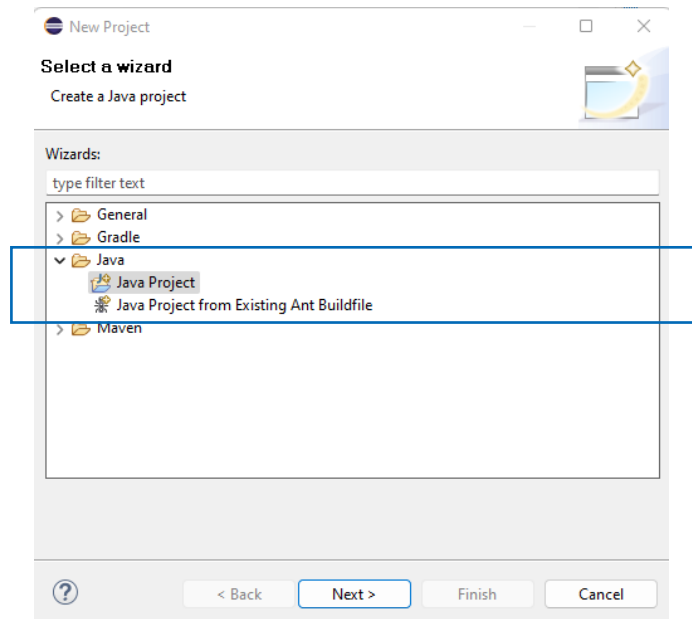
- About this course ([Update!! as of 12 Jan 2023](#))
- Object Oriented Programming
- Object -> Class (blueprint of object)
- Java environment

# Create Java Project in Eclipse

## 1. Click “File” Menu > New > Java Project



If you don't see “Java Project” option, select “Project” instead. In the “New Project” window, select “Java” folder, then “Java Project” and click “Next >” button.



## Create a Java Project

Enter a project name.



Project name:  2. Type the Project Name e.g., Lab01

☒ Use default location (Note that this is NOT a CLASS NAME)

Location:  C:\Users\MUICT\2022-2-ITCS209

JRE

☒ Use an execution environment JRE:  JavaSE-17

☐ Use a project specific JRE:  jre

☐ Use default JRE 'jre' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets

Working sets:

Module

☐ Create module-info.java file 3. Make sure that this box is “UNCHECKED”

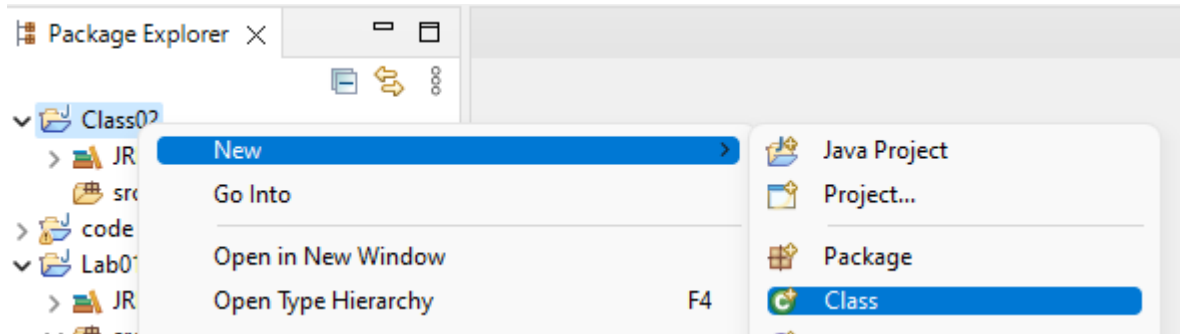
Module name:

☐ Generate comments

4. Click “Finish” button

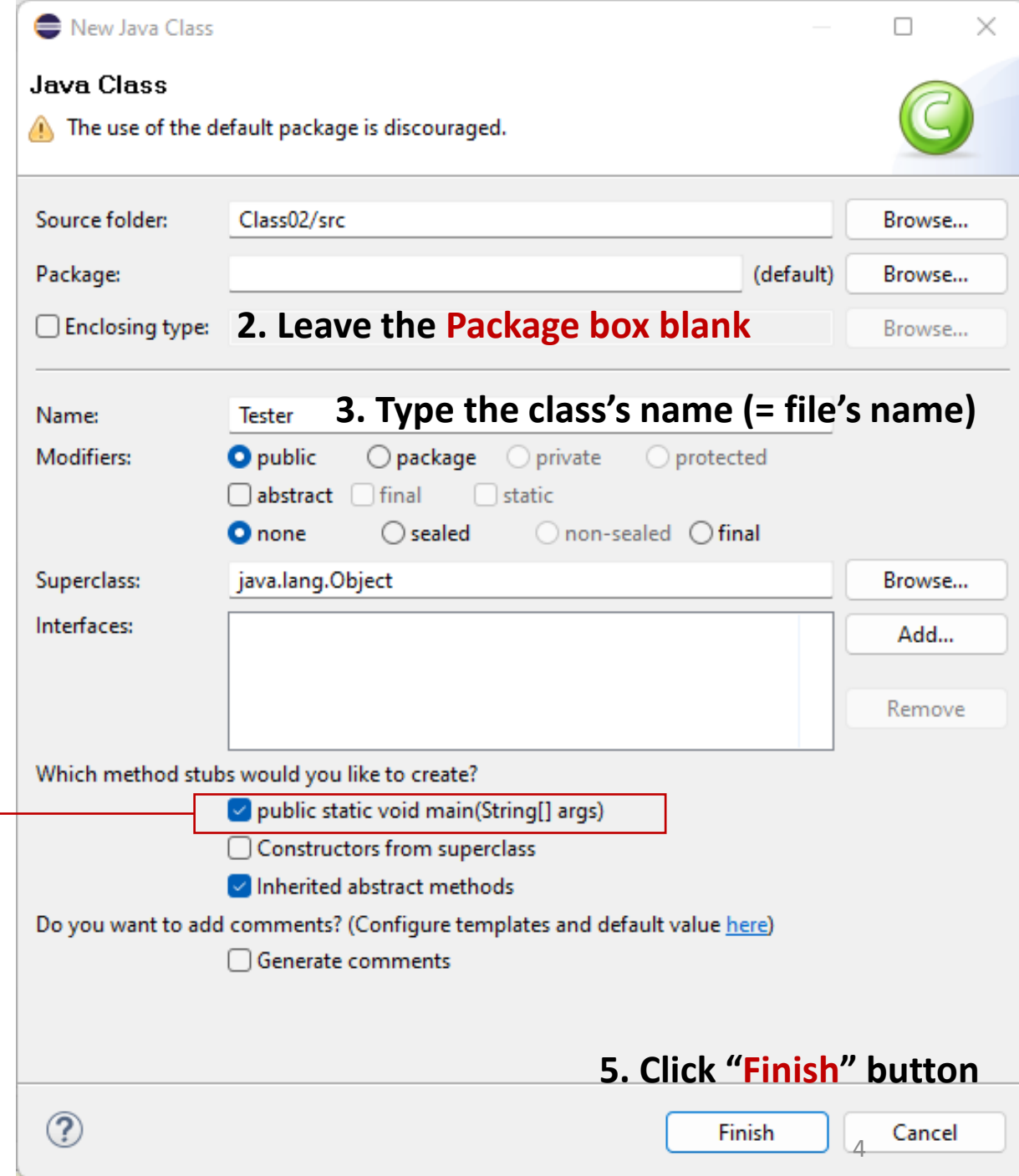
# Create Java Class in Eclipse

1. Right click at the java project, select New > Class



4. If this is the main class, select this box to include main method template. (optional)

```
1
2 public class Tester {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6     }
7
8
9 }
```





# Lab Assignment Submission

LAB

- Show your work with your TA or instructor & prepare to answer a few questions

My  
Courses

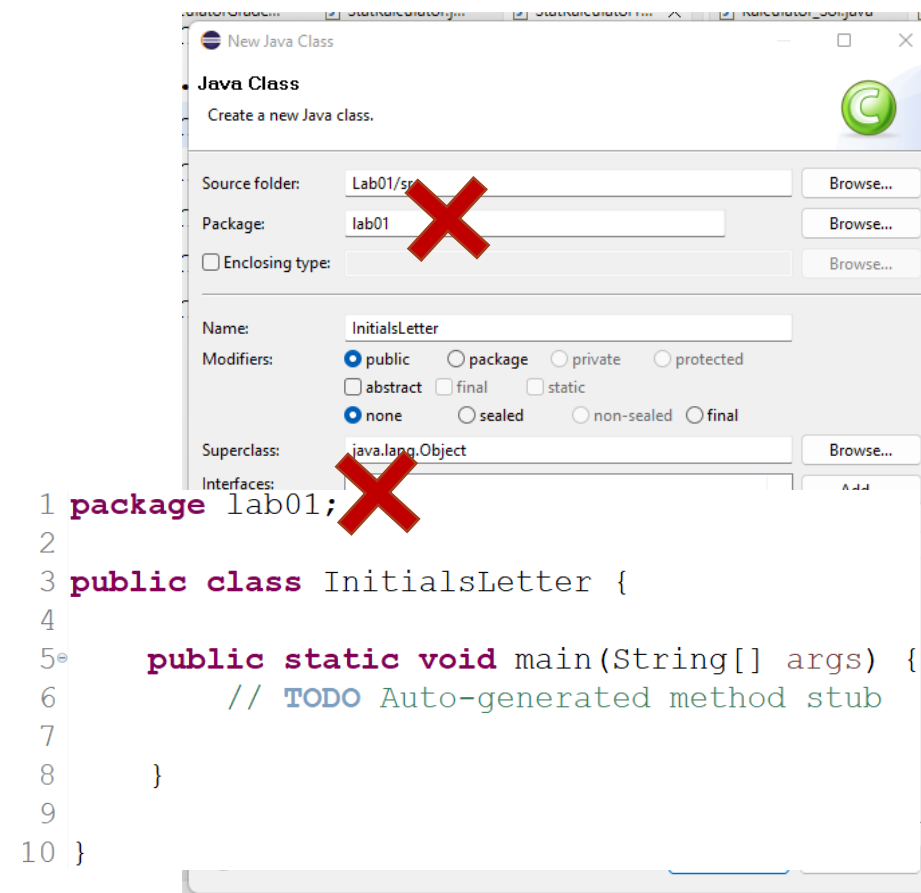
- Upload only **.java** file(s) on MyCourses before 6PM

# Common mistakes in last week's lab

- **DO NOT** have the header comment in your code

```
// File: <NAME OF FILE>
// Description: <A DESCRIPTION OF YOUR PROGRAM>
// Assignment Number: <e.g., 1 - 13>
//
// ID: <YOUR STUDENT ID>
// Name: <YOUR FULLNAME>
// Section: <YOUR SECTION e.g., 1, 2, or 3>
// Grader: <YOUR GRADER'S NAME>
//
// On my honor, <YOUR FULLNAME>, this lab assignment is my own work
// and I have not provided this code to any other students.
```

- **DO NOT** show your code and answer to LA
- **DO NOT** submit .java file in MyCourses
- Rename java file to be your name
- Java file contains **package** name



# Objective

- Variable
- Data Types
  - Primitive Type
  - Reference / Class Type
  - Constants
- Reading Input
- Decision
- Iteration



Assignment  
Comparison  
Arithmetic Operation  
Boolean Expression



# VARIABLE in JAVA



# Variables

- A **variables** is a storage location in a computer program. We can use a variable to store *values* or *objects* to be used later.



We need a **barcode** to identify each box in our big warehouse!!



# How to declare variable in JAVA?

```
// Syntax for variable declaration:
```

```
DataType variableName = value;
```

```
// or
```

```
DataType VariableName;
```

- **Data Type** is an attribute to describe data contained in a declared variable.
- Every variable in JAVA **must** specify its data type at declaration.
- This type **never** changes!

```
// Example
```

```
int age = 10;           // The variable age has type 'int' and the initial value is 10;
```

```
age = 22;               // The value 22 is an integer, so it can be stored in variable age
```

```
age = "Java"           // ERROR! The value Java is not an integer
```

# How to choose DataType?

- In practice, we will mostly use:

<code>boolean</code>	to represent <b>logic</b>	<b>primitive data type.</b>
<code>int, long and double</code>	to represent <b>numbers</b>	<b>primitive data type.</b>
<code>String</code> (chains of <code>char</code> )	to represent <b>text</b>	<b>Reference data type.</b>
<code>ObjectName</code>	to represent <b>object</b>	<b>Reference data type.</b>

e.g., `String text = "Welcome to Java";`

# Are these statements good 😊 or bad 😞

```
// 1  
int width = 10;
```

```
// 2  
int product price = 100;
```

```
// 3  
int qty = "3";
```

```
// 4  
int total = 3 * 100;
```

```
// 5  
String veggie = "Carrot"
```

```
// 6  
name = "Siripen";
```

```
// 7  
String msg = "Hello!";
```

```
// 8  
int a, b, c;
```

```
// 9  
int x = 10;  
int y = x;
```

```
// 10  
int i = 1, j = 2;  
String k = i + " is less than " + j;
```



# Names

- Variables names in Java have some restrictions. For example, `product price` cannot be a variable name.
- Here is a few rules for naming your variable:
  1. Start with an **English letter** or the **underscore ( \_ )**
  2. **No special symbols is allowed** except underscore.
  3. For multiple words, you can use “camelCases”.  
For example, the `product price` can be written as `productPrice`
  4. Java is case sensitive. `productPrice` is not the same as `productprice`
  5. Cannot use reserved words (such as `public`, `class`, `for`, `if`, etc.)





# DATA TYPE

## Primitive vs Reference Type

# Data Types

- In Java, every variable either
  - a reference to an **object** (class-types)

e.g. Car myCar = new Car();

String text = "String is an object";

- belongs to one of **8 primitive types**

e.g. **int** number = 25;

**boolean** check = true;

How to notice:

- Type starts with a capital letter:

**Car**

- Normal font style in Eclipse

How to notice:

- Type starts with a lowercase letter:

**int, boolean**

- Purple-bold** font style in Eclipse

Variable name	Value in memory
myCar	0x110
text	0x111
number	25
check	true

0x110



0x111

"String is  
an object"



# Eight Primitive Data Types

Table 1 Primitive Types		
Type	Description	Size
int	The integer type, with range −2,147,483,648 (Integer.MIN_VALUE) . . . 2,147,483,647 (Integer.MAX_VALUE, about 2.14 billion)	4 bytes
byte	The type describing a single byte, with range −128 . . . 127	1 byte
short	The short integer type, with range −32,768 . . . 32,767	2 bytes
long	The long integer type, with range −9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807	8 bytes
double	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
float	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
char	The character type, representing code units in the Unicode encoding scheme (see Computing & Society 4.2 on page 163)	2 bytes
boolean	The type with the two truth values false and true (see Chapter 5)	1 bit



# Number Types

- 4 integer types and 2 **floating point** types
- **int**: integers, no fractional part  
1, -4, 0
- **double**: floating-point numbers (double precision)  
0.5, -3.11111, 4.3E24, 1E-14
- A numeric computation **overflows** if the result falls outside the range for the number type

```
int n = 1000000;  
System.out.println(n * n); // prints 727379968
```

(e.g., max value of int is around 2.14 billions)

# Number Types

- **Rounding errors** occur when an exact conversion between numbers

*is not possible*

```
double f = 4.35;
```

```
System.out.println(100 * f);
```

```
// prints 434.99999999999994
```

to get perfect decimal precise  
use BigDecimal

- Java: **Legal** to assign an **integer value** to a **floating-point** variable

```
int dollars = 100;
```

```
double balance = dollars; // OK
```

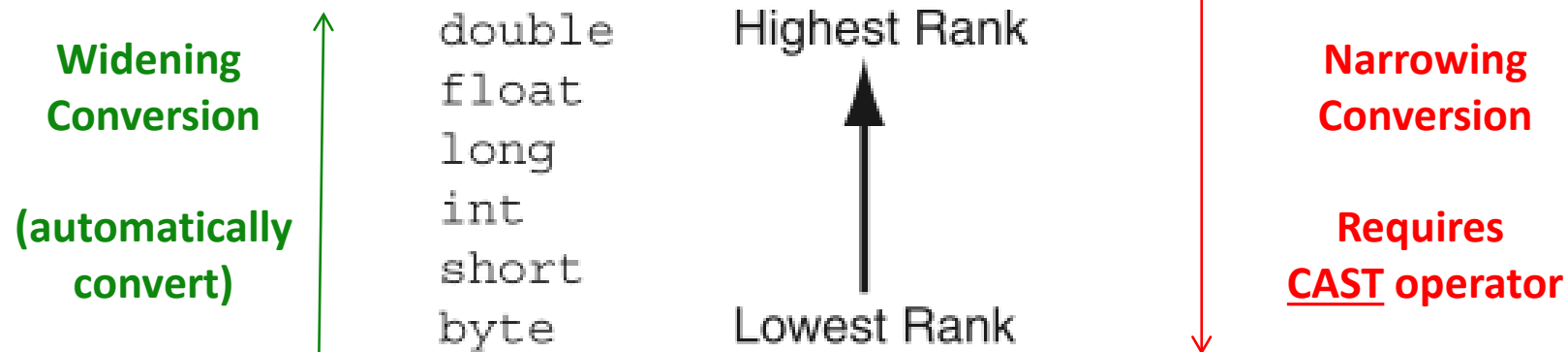
- Java: **Illegal** to assign a **floating-point** expression to an **integer** variable

```
double balance = 13.75;
```

```
int dollars = balance; // Error
```

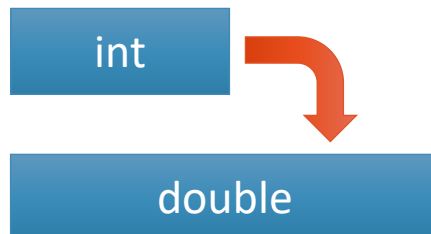
How to solve this problem?

# Casts

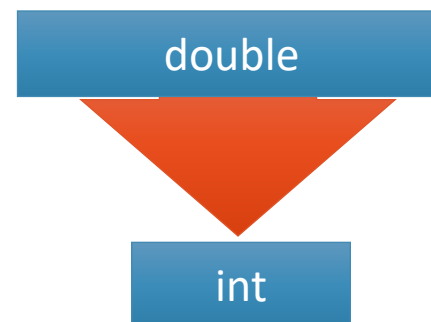


## Primitive Data Type Ranking

```
int y = 10;  
double x;  
x = y;
```



```
double x = 72.456  
int y;  
y = (int) x; // 72
```



# Casts

How about casting number types to String class type?

- Used to **convert** a value or an expression to a different type

- Synta `(typeName) expression;`

```
double balance = 13.75;  
int dollars = (int) balance;           // OK, dollars = 13  
int amountX = (int) balance * 100;     // amount = 1300  
int amountY = (int) (balance * 100);   // amount = 1375
```

Cast **discards** fractional part

- **Math.round** converts a floating-point number to nearest integer

```
long rounded = Math.round(balance);  
  
// if balance is 13.75, then rounded is set to 14  
  
// if balance is 13.5, then rounded is set to 14  
  
// if balance is 13.49, then rounded is set to 13
```

# boolean Data Type

- The **boolean** data type allows you to create variables that may hold one of two possible values: **true** or **false**.
- This data type is useful for evaluating **conditions**
- A *flag* is a boolean variable that signals when some condition exists in the program. – *we will talk more about this later*
- Unlike number types (e.g., **int**, **double**), the value of a boolean variable may NOT be casted to other primitive types

```
boolean bool;  
  
bool = true;  
System.out.println(bool);    // print: true  
  
bool = false;  
System.out.println(bool);    // print: false  
  
int test = (int) bool;        // ERROR!!
```

# Char Data Type 'x'

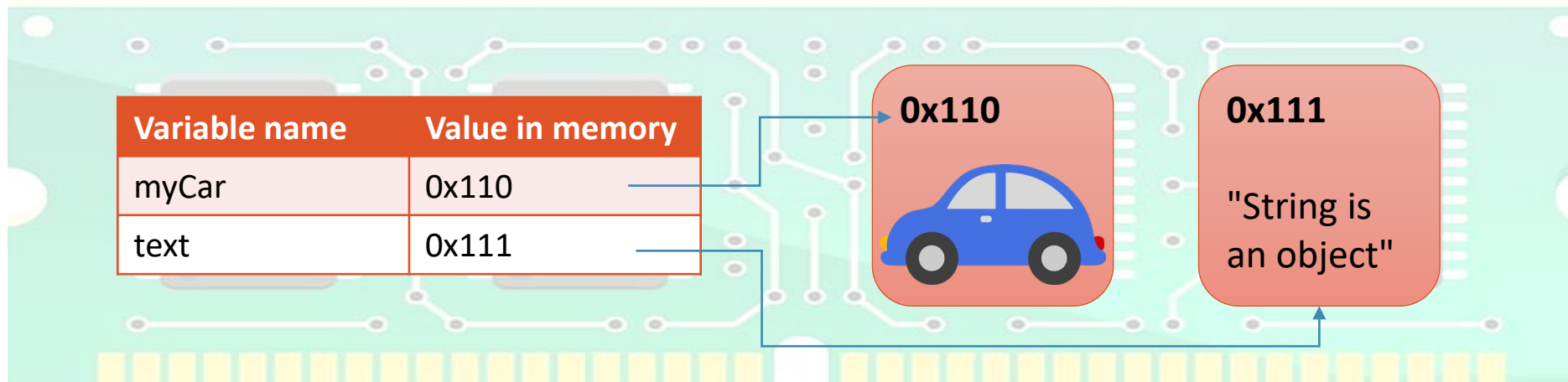
- A variable of the char data type can hold **ONE character** at a time
- Character literals are enclosed in *single quotation marks* ('')
- String literals **cannot** be assigned to char variables, and vice versa
- Characters are internally represented by **numbers**!

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	n1	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	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		

```
char ch = 'A';  
  
System.out.println(ch);           // A  
  
System.out.println((int) ch);     // 65  
  
System.out.println((double) ch); // 65.0  
  
ch = 66;  
  
System.out.println(ch);           // B
```

# Reference Data Types

**\* Variable stores an address to the actual value \***



# Reference Data Type

- **Reference Data Type** is a data types of a Class in which the value of variable contain reference to the actual value.
- There are two types of class
  - Pre-defined Class
    - Wrapper Classes (from 8 primitive types)
      - Integer, Byte, Short, Long, Double, Float, Character, Boolean
    - Other Classes
      - For example: String, ArrayList, ...
  - User-defined Class
    - For example: Car, Dog, Letter, ...



# Wrapper Classes

`Integer i = 3;`

`Character c = 'C';`

`Double d = 1.2;`

- Why do we need them?
  - Used with Collection object such as `ArrayList<Object>`
    - `ArrayList<Integer>` is correct but `ArrayList<int>` is error
  - Easily convert their value into String by calling `toString()` method

# Strings (1/3)

- `String` is a sequence of characters.

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

- Strings, though not of a primitive type, are frequently used
  - A variable created from the `String` class also count as an **object reference**.
- However, String is special. **It can be created directly** as primitive data type and **also create by using new keyword**
- Declaring String:
- Initializing string with empty string:

```
String message1 = "ABC";
```

```
String message2 = new String ("XYZ");
```

```
String message3 = ""; OR
```

```
String message4 = new String();
```

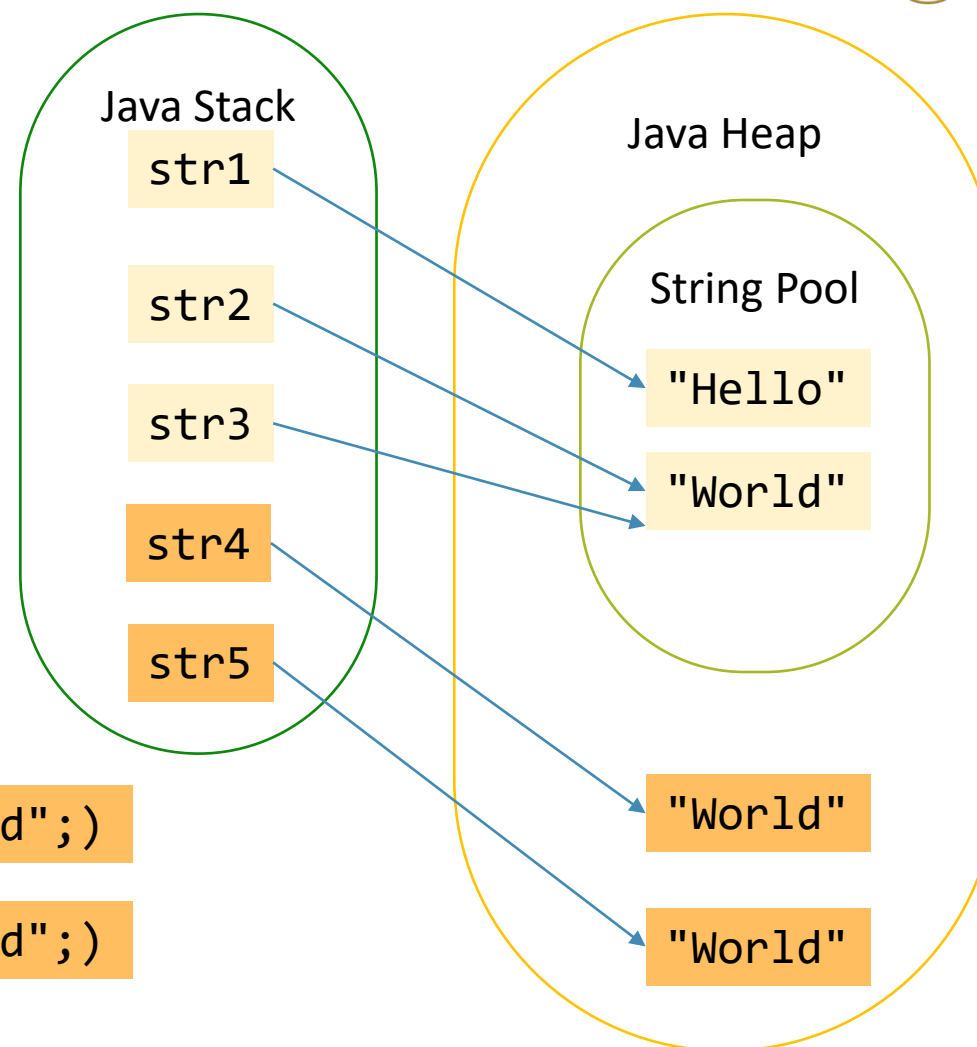
```
String str1 = "Hello";
```

```
String str2 = "World";
```

```
String str3 = "World";
```

```
String str4 = new String("World");
```

```
String str5 = new String("World");
```



**NOTE**

```
str2 == str3      // true
```

```
str4 == str5      // false
```

## Strings (2/3)

- Since `String` is a class, it contains several attributes/fields, constructors and **methods**

→ You may search for `String` API for more detail

Statement	Result	Comment
<pre>string str = "Ja"; str = str + "va";</pre>	str is set to "Java"	When applied to strings, + denotes concatenation.
<pre>System.out.println("Please" + " enter your name: ");</pre>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<pre>team = 49 + "ers"</pre>	team is set to "49ers"	Because "ers" is a string, 49 is converted to a string.
<pre>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</pre>	first contains "Harry" last contains "Morgan"	The next method places the next word into the string variable.
<pre>String greeting = "H &amp; S"; int n = greeting.length();</pre>	n is set to 5	Each space counts as one character.

# Strings (3/3)

Statement	Result	Comment
<pre>String str = "Sally"; char ch = str.charAt(1);</pre>	ch is set to 'a'	This is a char value, not a String. Note that the initial position is 0.
<pre>String str = "Sally"; String str2 = str.substring(1, 4);</pre>	str2 is set to "all"	Extracts the substring starting at position 1 and ending before position 4.
<pre>String str = "Sally"; String str2 = str.substring(1);</pre>	str2 is set to "ally"	If you omit the end position, all characters from the position until the end of the string are included.
<pre>String str = "Sally"; String str2 = str.substring(1, 2);</pre>	str2 is set to "a"	Extracts a String of length 1; contrast with <code>str.charAt(1)</code> .
<pre>String last = str.substring(     str.length() - 1);</pre>	last is set to the string containing the last character in str	The last character has position <code>str.length() - 1</code> .

# Example

```
String message = "Java is Great Fun!";

String upper = message.toUpperCase();    // JAVA IS GREAT FUN!

char letter = message.charAt(2);          // v

String sub2 = message.substring(5, 10);    // is Gr

String sub3 = message.substring(10, 5);    // ERROR!

String lastTwoChar = message.substring (message.length()-2); // n!

String sub4 = message.substring (8, message.length()-2); //Great Fu
```

J	a	v	a		i	s		G	r	e	a	t		F	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Index start with 0

# Strings Concatenation

- Use the + operator:

```
String name = "Dave";  
String message = "Hello, " + name;  
// message is "Hello, Dave"
```

- If one of the arguments of the + operator is a string, the other is converted to a string

```
String a = "Agent";  
int n = 7;  
String bond = a + n; // bond is "Agent7"
```

- Reduce print statements

```
System.out.print("The total is ");  
System.out.println(total);
```

} System.out.print("The total is " + total);

# Strings Conversion

- Convert to number:

```
String str = "19";  
int n = Integer.parseInt(str);
```

```
String str2 = "3.95";  
double x = Double.parseDouble(str2);
```

- Convert to string:

```
String str = "" + n;
```

-----OR-----

```
String str = Integer.toString(n);
```



# Self Check

String message = "Java is Great Fun!";

String upper = message.toUpperCase();

char letter = message.charAt(2);

String sub2 = message.substring(5, 10);

String sub3 = message.substring(10, 5);

String lastTwoChar = message.substring (message.length()-2);

String sub4 = message.substring (8, message.length()-2);

J	a	v	a		i	s		G	r	e	a	t		F	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Index start with 0



# CONSTANT

# Constants

- **Constant** is a variable that once its value has been set, it *cannot be changed*.
- Using constants makes your program easier to maintain and read.
- Ex. Use `PI` instead of  
3.141592653589793238462643383279502884197169399375105820974944592307816406286
- **Usage:** a keyword **final** is used in front of data type
- E.g., `public final double PI = 3.14159265358979323;`

# Constant with Static

- The **static** constants allow other classes to use them directly
- Example **Math** class

```
public class Math {  
    . . .  
    public static final double E = 2.718281828459045235;  
    public static final double PI = 3.14159265358979323;  
}
```

- This constant can be referred in other class as:

```
double circumference = Math.PI * diameter;
```

# Enum

- If you want to define your own variable with specific set of possible values, you can use Enum
  - For example, Days of week, Gender, Color

```
public enum Color {Black, Blue, Green, Red}  
  
public static void main(String[] args) {  
    Color shirt = Color.Black;  
    Color skirt = Color.Yellow;  
}
```



# ASSIGNMENT (=) & COMPARISION (==)

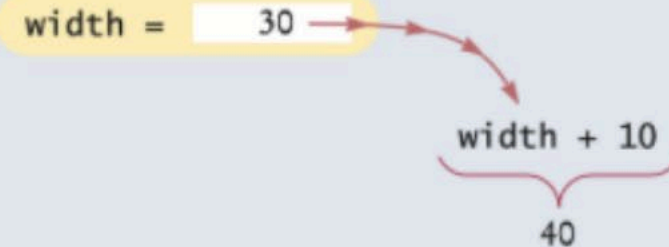
# Assignment

- **Assignment:** a statement to assign the result of an expression (on the right) to the variable (on the left)

`variable = expression;`

```
// 1  
int width = 30;  
  
// 2  
width = width + 10;
```

- 1 Compute the value of the right-hand side



- 2 Store the value in the variable



width = 40

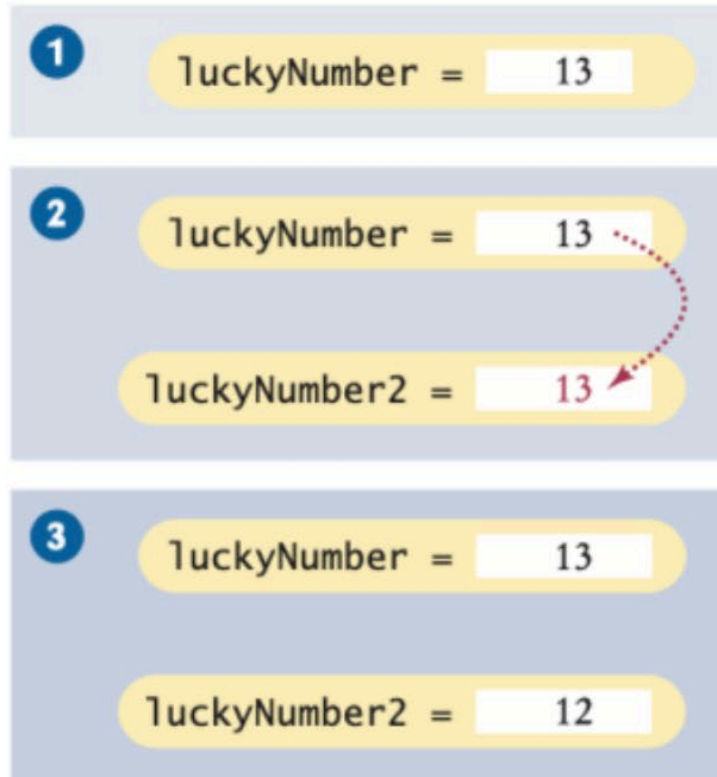
# Assignment

- The value is copied from one variable to the other when you assign

```
// 1
int luckyNumber = 13;

// 2
int luckyNumber2 = luckyNumber;

// 3
luckyNumber2 = 12;
```





# Comparing NUMBERS

Relational operators are used to compare numeric values.

```
int a = 5;  
int b = 10;
```

Operator	Meaning	Example	Result
<	Less than	if ( a < b )	True
<=	Less than or equal to	if ( a <= b )	True
>	Greater than	if ( a > b )	False
>=	Greater than or equal to	if ( a >= b )	False
==	Equal	if ( a == b )	False
!=	Not equal	if ( a != b )	True

Return either True or False

# = VS ==

- Be careful and Don't confuse between = and ==
  - The == operator tests for equality

```
x == 0          // return true if x is 0,  
                // otherwise false
```

- usually use in if-else condition and loop
- e.g., if(x == 0)

```
    System.out.println("x is 0");
```

- The = operator assigns a value to a variable

```
x = 0;          // assign 0 to x
```

# Comparing STRINGS

- Do **NOT** use `==` for Strings!

Check the address!

```
String input = new String("Y");  
if (input == "Y") {...}           // Always FALSE!!!
```



- Use **equals** method:

```
if (input.equals("Y")) {...}      // true
```

Check the content!

- Note! Case sensitive test (`"Y" != "y"`), to ignore the letter case use `equalsIgnoreCase` method

```
if (input.equalsIgnoreCase("y")) {...}      // true
```

# Testing for null

- An object reference can have the special value `null` if it refers to no object at all.
- using `obj == null` OR `obj != null`

To avoid an error

```
String str = null;  
System.out.println(str.length());    // Throw exception at Runtime!
```

```
if(str != null)  
    System.out.println("The string length is " + str.length());  
else  
    System.out.println("This string is null");
```

```
String str = "";  
System.out.println(str.length());    // prints 0
```

- Note that the `null` reference is not the same as the `empty string` `""`.
- The `empty string` is a valid string of length 0, whereas a `null` indicates that a string variable refers to no string at all.

# Logical Operations

- **Boolean expression:** a logical statement that either **true** or **false**.
- You can combine boolean expressions with logical operators.
- For example,

```
(vaccine == true) && (age > 11)
```

Operand	Example	Meaning
&&	var1 && var2	Are both values true?
	var1    var2	Is at least one value true?
!	!var1	Is it NOT var1?

# Boolean Expression: Logical Ops

Operator	Meaning	Effect
&&	AND	Connects two boolean expressions into one. Both expressions must be true for the overall expression to be true.
	OR	Connects two boolean expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	NOT	The ! operator reverses the truth of a boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

A	B	A    B	A && B	!A
0	0	0	0	1
1	0	1	0	0
0	1	1	0	1
1	1	1	1	0

```
if( score > 70 && score <= 80)  
    grade = 'B';
```

```
if( x == 10 || y == 20)
```

```
if( !(x > y) )
```



# ARITHMETIC OPERATIONS

# Arithmetic Operations

Operator	Meaning	Type	Example
+	Addition	Binary	<code>total = cost + tax;</code>
-	Subtraction	Binary	<code>cost = total - tax;</code>
*	Multiplication	Binary	<code>tax = cost * rate;</code>
/	Division	Binary	<code>salePrice = original / 2;</code>
%	Modulus	Binary	<code>remainder = value % 3;</code>
-	Negation	Unary	<code>x = -5;</code>

## // Example

```
int amount = 4 + 8;           // 12
amount = 4 - 8;               // -4
amount = 5 * 20;              // 100
amount = 100 / 5;             // 20
int remainder = 10 % 9; // 1
```

// Try again

```
int x = 5, y = 20;
```

```
x = x + 1;           // x = 6
```

```
y = y - x;           // y = 14
```

```
x = y % x;           // x = 2
```

```
x = x / 100;         //
```



# Integer Division (/)

```
int number;  
number = 5 / 2; // 2
```

```
double num2;  
num2 = 5 / 2;    // 2.0
```

```
num2 = 5.0 / 2.0; // 2.5
```

```
num2 = 5.0 / 2;  // 2.5
```

```
num2 = 5 / 2.0;  // 2.5
```

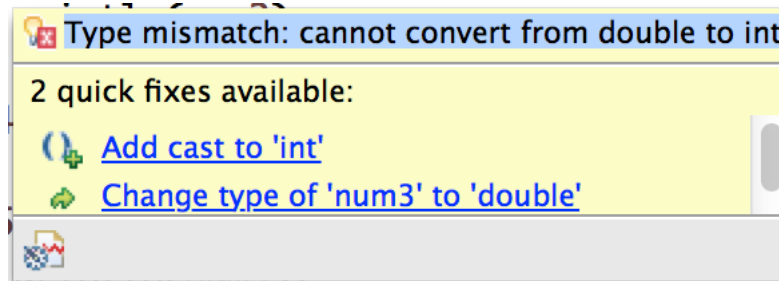
```
int num3;  
num3 = 5.0 / 2;  // ERROR!!
```

Since the number 5 and 2 are both integers, the fractional part of the result will be discarded before the assignment takes place.

**BE CAREFUL!**

```
double num4 = 5.0 / 0;  
// OK, no syntax error (^_^) yeah
```

```
System.out.println(num4);  
// java.lang.ArithmeticException: / by zero (T-T)
```



# Operators Precedence (1/3)

- Building mathematical expressions with several operators!

$a = b + c * d$



**Expression**  
(variable + operand)

Operator  
Operand

if  $x = 2$  and  $y = 20$ , what are the answers of these expressions?

1.  $\text{answer} = 10 + x + 5 + y$       //  $\text{answer} = 37$

2.  $\text{answer} = y / x + 5$       //  $\text{answer} = 15$

3.  $\text{answer} = 12 + x / 2$       //  $\text{answer} = 13$

4.  $\text{answer} = (12 + x) / 2$       //  $\text{answer} = 7$

# Operators Precedence (2/3)

Operator	Precedence
( )	High ↑ Low
Unary Operator (-)	
*, /, %	
+, -	

answer = 12 + 2 / 2

answer = 12 + 1

answer = 13

# Self Check

$$2 * (-5 - (17 \% 3 / 2) + 26) * 2 + 4 = ?$$



# The Math Class

- **Java API** provides a class named Math, which contain numerous methods for performing complex mathematical operations.
- <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

- *Example:*

```
Math.round → Math.round(12.5)    // 13
Math.abs   → Math.abs(-5);        // 5
Math.pow   → Math.pow(2.0, 3.0)    // 23 = 8
Math.sqrt  → Math.sqrt(25.0)      // 5
```

- **Note:** to compute  $x^2$ , it is more practical to simply use `x*x`;
- How to write this formula in java?

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

# The Math Class

$$\begin{aligned} & (-b + \text{Math.sqrt}(b * b - 4 * a * c)) / (2 * a) \\ & \quad \underbrace{\quad \quad \quad}_{b^2} \quad \underbrace{\quad \quad \quad}_{4ac} \quad \underbrace{\quad \quad}_{2a} \\ & \quad \quad \quad \underbrace{\quad \quad \quad}_{b^2 - 4ac} \\ & \quad \quad \quad \underbrace{\quad \quad \quad}_{\sqrt{b^2 - 4ac}} \\ & \quad \quad \quad \underbrace{\quad \quad \quad}_{-b + \sqrt{b^2 - 4ac}} \\ & \quad \quad \quad \underbrace{\quad \quad \quad}_{\frac{-b + \sqrt{b^2 - 4ac}}{2a}} \end{aligned}$$

# Self Check

- What is the value of  $1729 / 100$ ? and  $1729 \% 100$ ?



- How to find the average of 3 integer values?

`double average = s1 + s2 + s3 / 3; // Is this one correct?`

- What is the value of the following statement in mathematic notation?

`Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))`

# Combined Assignment Operators

- The combined assignment operators combine the assignment operator with the arithmetic operators.
- Also known as ***compound operators***

Operator	Example Usage	Equivalent to
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>

Can compound operators be used with *String* variable?

```
String str = "This is ";  
str += "a pen.";  
System.out.println(str);  
// This is a pen.
```



# Increment (++) and Decrement (--)

- Increment and decrement operator are **unary** operator
- **Unary** Operator Operates on One Operand.
- **Increment Operator (++)** is used to increment value stored inside variable on which it is operating by 1.
  - E.g., num++, ++num (post and pre increment)
- **Decrement Operator (--)** is used to decrement value stored inside variable by 1.
  - E.g., num--, --num (post and pre decrement)

# Example

- What is the value of *n* in the first and second *println* method?

```
int n = 0;
```

```
System.out.println(n++); // 0
```

```
System.out.println(n);    // 1
```

```
System.out.println(n);  
n = n + 1;  
System.out.println(n);
```

- What is the value of *n* in the first and second *println* method?

```
int m = 0;
```

```
System.out.println(++m); // 1
```

```
System.out.println(m);    // 1
```

```
m = m + 1;  
System.out.println(m);  
System.out.println(m);
```

# DECISION

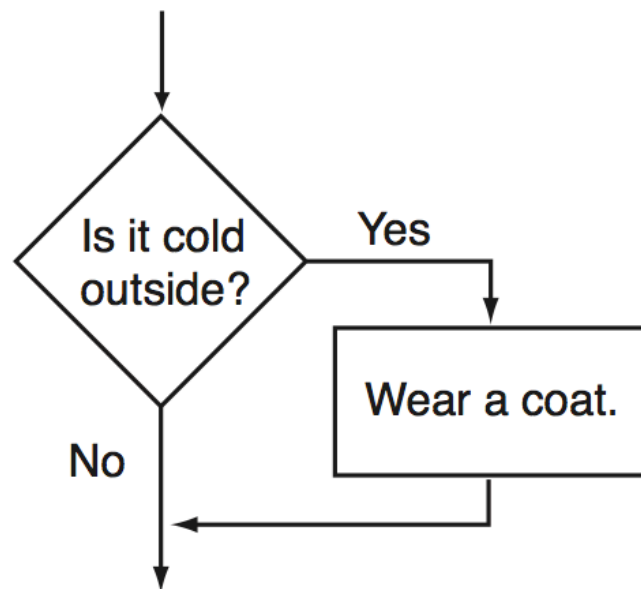
if/else statement,  
multiple alternative,  
switch statement



# if Statement

- The if statement is used to create a decision structure, which allows a program to have more than one path of execution.
- The if statement causes *one or more* statements to execute only when a *boolean expression* is *true*.

## Simple decision structure logic



```
if (BooleanExpression)  
    statement;
```

This action is **conditionally executed** because it is performed only when a certain condition (cold outside) exists.

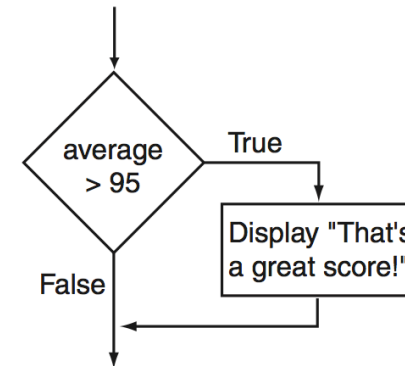
# Example 1: Single Statement



No semicolon (;) here

```
if (average > 95) {  
    System.out.println("That's a great score!");  
}
```

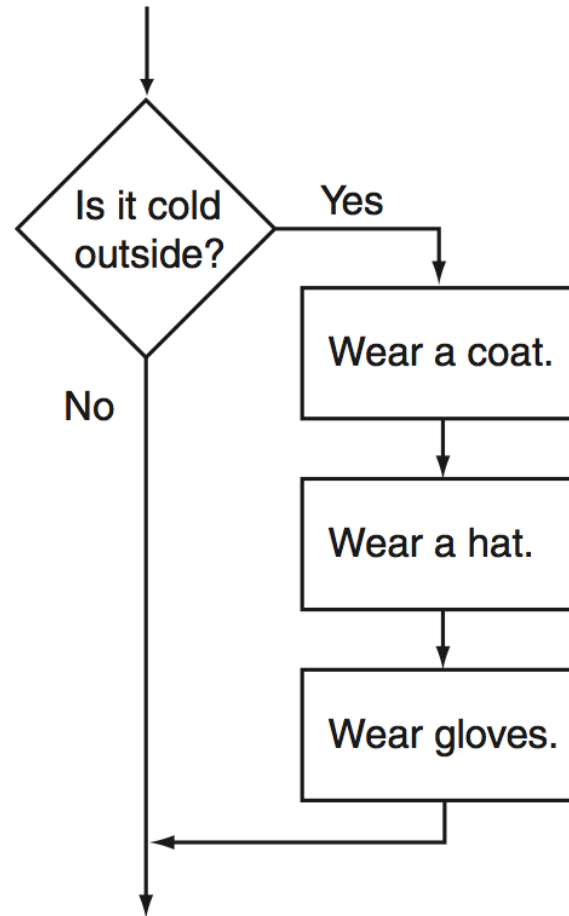
Note: If there is only 1 conditional executed statement, you may ignore the `{ }`



Statement	Outcome
<pre>if (hours &gt; 40)     overTime = true;</pre>	If hours is greater than 40, assigns true to the boolean variable overTime.
<pre>if (value &lt; 32)     System.out.println("Invalid number");</pre>	If value is less than 32, displays the message "Invalid number"

# if Statement

## Three-action decision structure logic



```
if (BooleanExpression){  
    statement_1;  
    statement_2;  
    statement_3;  
    ...  
    statement_n;  
}
```

## Example 2: Block of Statements

Enclosing a group of statements inside braces `{ }` creates a *block* of statements.

```
if (sales > 50000) {  
    bonus = 500.0;  
    commissionRate = 0.12;  
    daysOff += 1;  
}
```

*An If statement missing its braces*

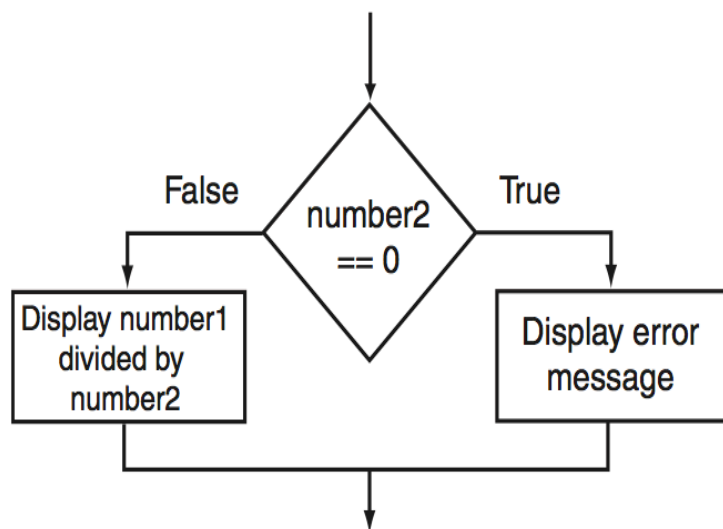
```
if (sales > 50000)  
    bonus = 500.0;  
    commissionRate = 0.12;  
    daysOff += 1;
```

Only this statement is conditionally executed.

These statements are *always* executed.

# if-else statement

- The if-else statement is an expansion of if statement
- It will execute one group of statements if its boolean expression is true, or another group if its boolean expression is false.

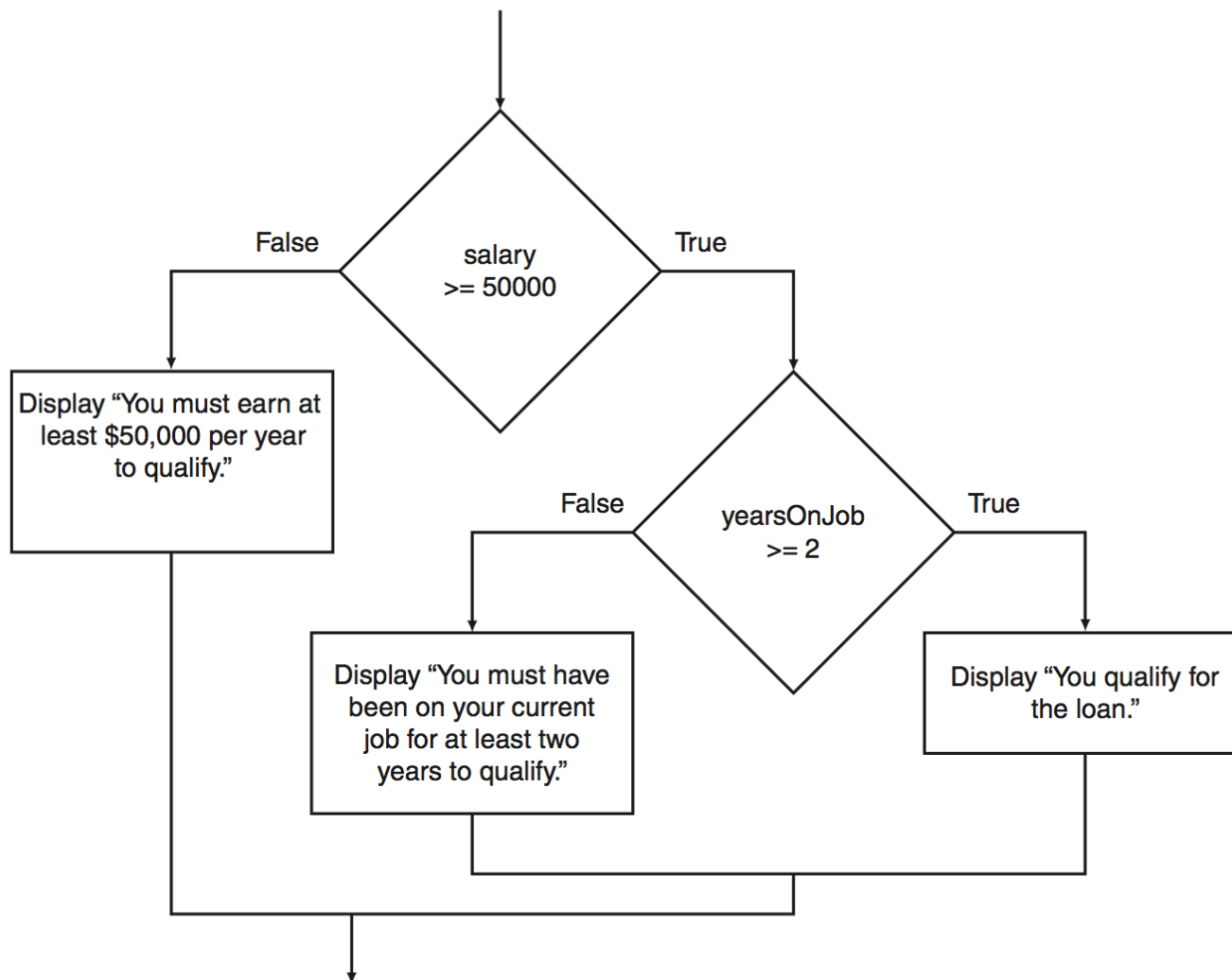


```
if (BooleanExpression)
    statement or block;
else
    statement or block;
```

```
if (number2 == 0){
    System.out.println("Number1 cannot
        be divided by 0.");
} else{
    System.out.println(number1/number2);
}
```

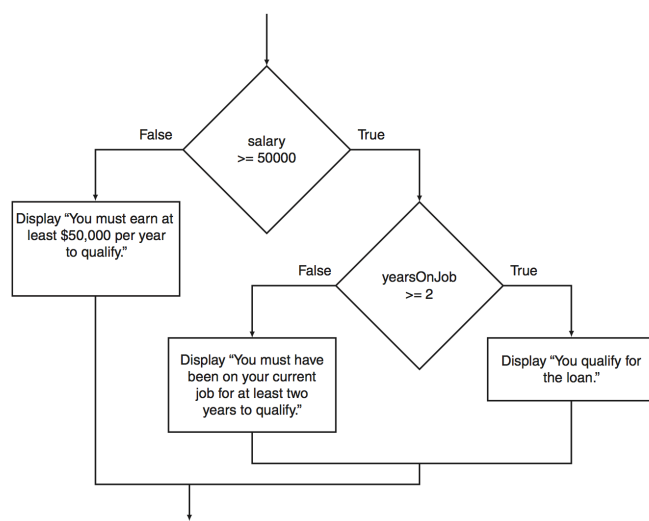
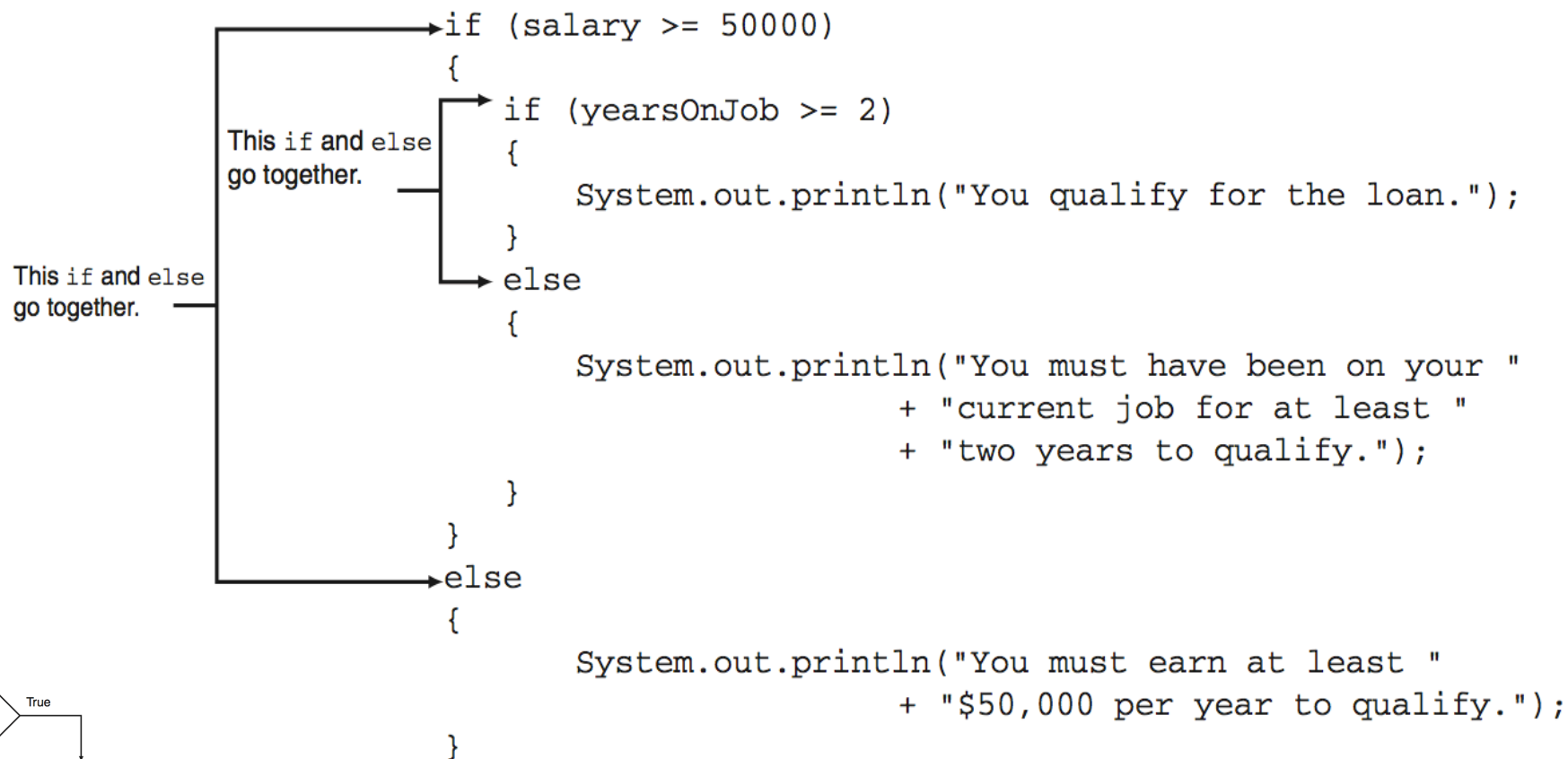


# Nested if-else Statement



```
if (booleanExpression1) {  
    if(booleanExpression2)  
        statement or block;  
    else  
        statement or block;  
}  
else  
    statement or block;
```

# Nested if-else Statement



# if-else-if Statement

The if-else-if statement tests a series of conditions.

It is often **simpler** to test a series of conditions with the if-else-if statement than with a set of **nested** if-else statements.

```
if (expression_1)
{
    statement
    statement
    etc.
}
```

*If expression\_1 is true these statements are executed, and the rest of the structure is ignored.*

```
else if (expression_2)
{
    statement
    statement
    etc.
}
```

*Otherwise, if expression\_2 is true these statements are executed, and the rest of the structure is ignored.*

*Insert as many else if clauses as necessary*

```
else Do not omit 'else'
{
    statement
    statement
    etc.
}
```

*These statements are executed if none of the expressions above are true.*

# Nested If-Else vs. If-Else-If Statement

```
char grade;

if (score < 60)
{
    grade = 'F';
}
else
{
    if (score < 70)
    {
        grade = 'D';
    }
    else
    {
        if (score < 80)
        {
            grade = 'C';
        }
        else
        {
            if (score < 90)
            {
                grade = 'B';
            }
            else
            {
                grade = 'A';
            }
        }
    }
}
}
```

*If the score is less than 60, then the grade is F.  
Otherwise, if the score is less than 70, then the grade is D.  
Otherwise, if the score is less than 80, then the grade is C.  
Otherwise, if the score is less than 90, then the grade is B.  
Otherwise, the grade is A.*

```
char grade;

if (score < 60)
    grade = 'F';
else if (score < 70)
    grade = 'D';
else if (score < 80)
    grade = 'C';
else if (score < 90)
    grade = 'B';
else
    grade = 'A';
```

# Exercise – Drawing a flowchart



- Print “happy” if the happy level is larger than 70



- Print “happy” if the happy level is larger than 70, otherwise print “sad”.

- If the given number is even, print whether the number is divisible by 4 or not. If the number is odd, print whether the number is divisible by 3 or not.

- Check and print whether the given year is a leap year or not. Leap year is the year that is an integer multiple of 4 (except for years divisible by 100, but not by 400).

# Using boolean variables as Flags

- A *flag* is a boolean variable that signals when some condition exists in the program.
- For example, the program has a boolean variable named `done`. This variable is used to signal that the program is finished when the score is higher than or equal to 3

```
if (score > 3)
    done = true;
```

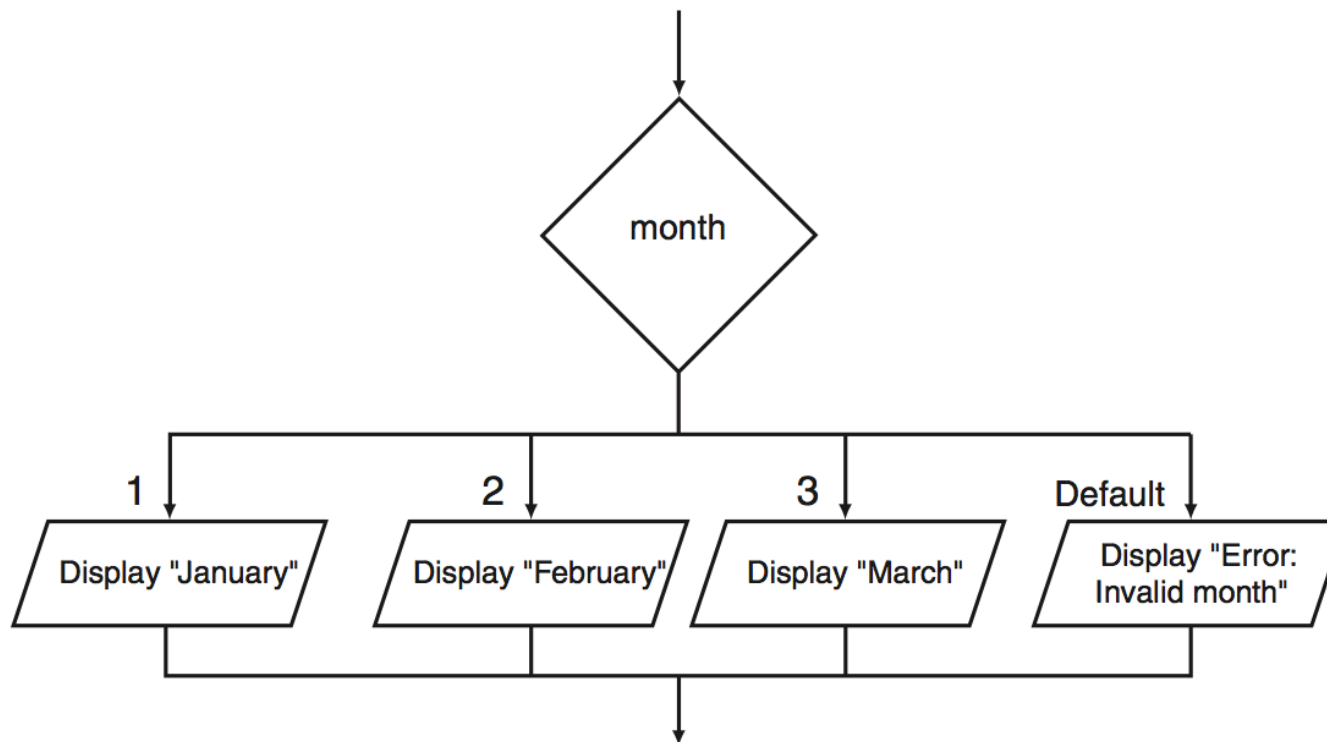
- Later, you may refer to this *flag* variable

```
if (done)
    System.out.println("Great Job! You have completed the lab assignment.
                        You may leave the room.");
```

- You will find flag variables useful in many circumstances, and we will come back to them in future chapters.

# switch Statement

- It also use to create *multiple alternative decision structure* by allowing a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each case.



The testExpression is  
a variable or expression.

that gives a char, byte, short, int, or string value. (NOT BOOLEAN)

```
switch (testExpression)
{
```

```
    case value_1:
        statement;
        statement;
        etc.
        break;
```

These statements are executed  
if the testExpression is  
equal to value\_1.

```
    case value_2:
        statement;
        statement;
        etc.
        break;
```

These statements are executed  
if the testExpression is  
equal to value\_2.

Insert as many case sections as necessary.

```
    case value_N:
        statement;
        statement;
        etc.
        break;
```

These statements are executed  
if the testExpression is  
equal to value\_N.

default: (Optional)

```
    statement;
    statement;
    etc.
    break;
```

These statements are executed  
if the testExpression is not  
equal to any of the case values.

```
}
```

```
switch (month) {
    case 1:
        System.out.println("Jan");
        break;
    case 2:
        System.out.println("Feb");
        break;
    case 3:
        System.out.println("Mar");
        break;
    default:
        System.out.println("---");
        break;
}
```

```
if (month == 1)
    System.out.println("Jan");
else if (month == 2)
    System.out.println("Feb");
else if (month == 3)
    System.out.println("Mar");
else
    System.out.println("---");
```



# switch Statement (without **break;**)

```
input = keyboard.nextLine();
feedGrade = input.charAt(0); // Get the first char.

// Determine the grade that was entered.
switch(feedGrade)
{
    case 'a':
    case 'A':
        System.out.println("30 cents per lb.");
        break;
    case 'b':
    case 'B':
        System.out.println("20 cents per lb.");
        break;
    case 'c':
    case 'C':
        System.out.println("15 cents per lb.");
        break;
    default:
        System.out.println("Invalid choice.");
}
```

When the user enters 'a' the corresponding case has no statements associated with it, so the program *falls through* to the next case, which corresponds with 'A'. Same technique for 'b' and 'c'.

So, if the input for a user is

A [Enter]

Output: 30 cents per lb.

B [Enter]

Output: 20 cents per lb.

b [Enter]

Output: 20 cents per lb.

c [Enter]

Output: 15 cents per lb.

d [Enter]

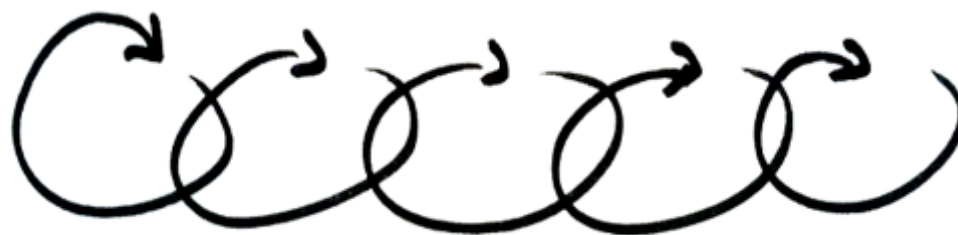
Output: Invalid choice.

# Self Check

What is wrong with the following switch statement?

```
// This code has errors!!!  
switch (temp)  
{  
    case temp < 0 :  
        System.out.println("Temp is negative.");  
        break;  
    case temp == 0 :  
        System.out.println("Temp is zero.");  
        break;  
    case temp > 0 :  
        System.out.println("Temp is positive.");  
        break;  
}
```





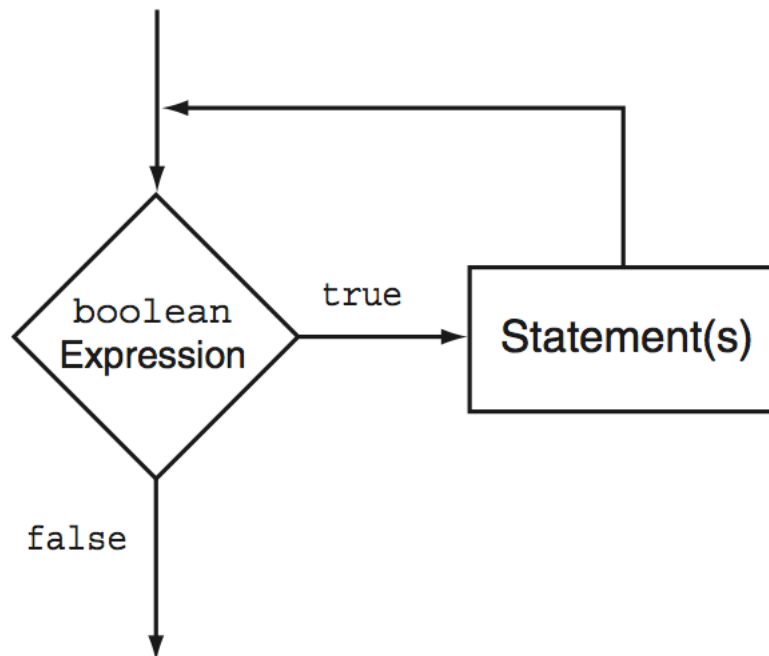
# Iteration

- **While Loop**
- **For Loop**
- **Do Loop**



# while Loop (1/2)

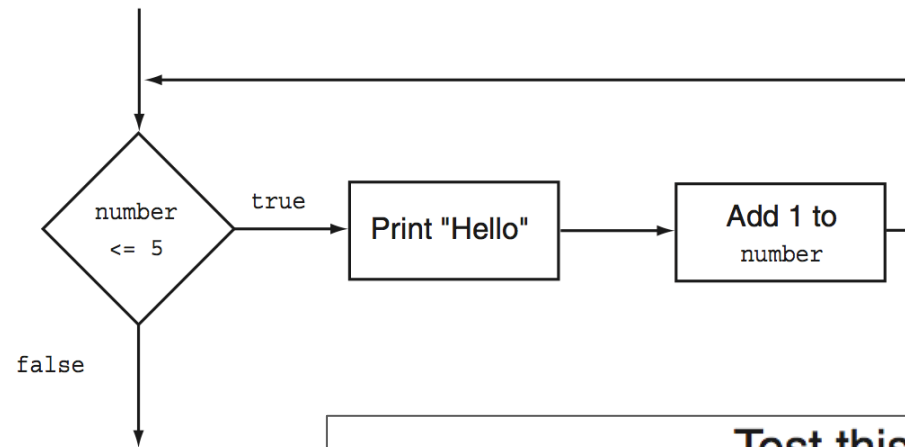
- While loop: a statement for repeatedly executes a body statement as long as a given condition is true
- Pretest loop



```
while (BooleanExpression)
    statement;
```

```
while (BooleanExpression) {
    statement;
    statement;
    // Place as many statements
    // here as necessary.
}
```

# while Loop (2/2)



Test this boolean expression.

```
while (number <= 5)
{
    System.out.println("Hello");
    number++;
}
```

If the boolean expression  
is true, perform these statements.

After executing the body of the loop, start over.

# Example 1

- Print **even** number from 0 to 8

```
int i = 0;
while (i < 5) {
    System.out.println(i * 2);
}
/* Output:
0
2
4
6
8
*/
```

## Example 2

- If you deposit \$10,000 with an interest of 5%, how many year until you have more than \$20,000

```
double balance = 10000;
double targetBalance = 20000;
int years = 0;
while (balance < targetBalance) {
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
System.out.println(years);
/* Output:
15
*/
```

Tracing:

- 1st iteration: `balance` : 10000 < `targetBalance` : 20000 ✓  
`years` : 1  
`balance` : 10500
- 2nd iteration: `balance` : 10500 < `targetBalance` : 20000 ✓  
`years` : 2  
`balance` : 11025
- ...
- 16th iteration `balance` : 20789.28 < `targetBalance` : 20000 ✗  
stops the loop



# Self Check

- How many times will "Hello World" be printed in the following program segment?

```
int count = 10;  
while (count < 1) {  
    System.out.println("Hello World");  
    count++;  
}
```

Answer:



- How many times will "I love Java programming!" be printed in the following program segment?

```
int count = 0;  
while (count > 10)  
    System.out.println("Hello World");  
    System.out.println("I love Java programming!");
```

Answer.

# for Loop

- **For Loop:** an execution of a body statements in a specific number of times.
- It is useful when you know how many times a task is to be repeated.

```
for (initialization; condition; update)  
    statement;
```

```
for (initialization; condition; update) {  
    statement;  
    statement;  
    // Place as many statements here  
    // as necessary.  
}
```

# for Loop



**Step 1:** Perform the initialization expression.

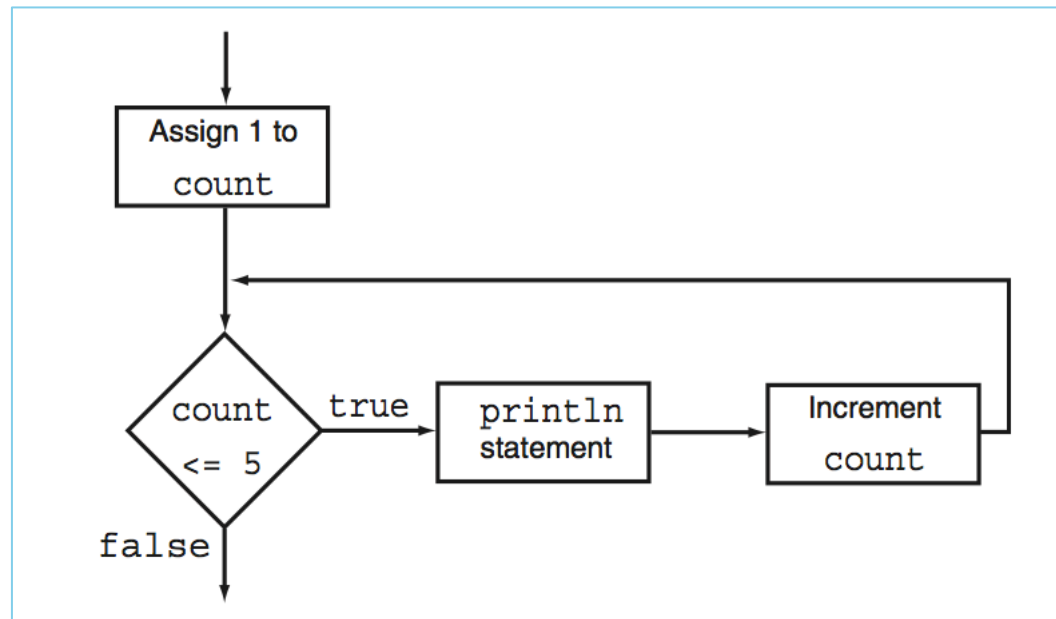
**Step 2:** Evaluate the test expression. If it is true, go to step 3. Otherwise, terminate the loop.

```
for (count = 1; count <= 5; count++)
```

```
System.out.println("Hello");
```

**Step 3:** Execute the body of the loop.

**Step 4:** Perform the update expression, then go back to step 2.



# Examples



```
int number;  
for (number = 1; number <= 10; number++) {  
    System.out.print(number + " ");  
}
```

Output: 1 2 3 4 5 6 7 8 9 10

```
for (int number = 1; number <= 10; number++) {  
    System.out.print(number + " ");  
}
```

Output: 1 2 3 4 5 6 7 8 9 10

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

Output: Display **even numbers**  
from 2 through 100

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

Output: Compilation Fails

# Examples

- If you deposit \$10,000 with an interest of 5%, how much money you will have after 15 years?

```
double balance = 10000;
int numberOfYears = 15;
for (int i = 0; i < numberOfYears; i++)
{
    double interest = balance * rate / 100;
    balance = balance + interest;
}
System.out.println(balance)
/* Output:
20789.28
*/
```

Tracing:

- 1st iteration: `i` : 0 < `numberOfYears` : 15 ☒  
`balance` : 10500  
`i` : 1
- ...
- 16th iteration: `i` : 15 < `numberOfYears` : 15 ☐  
stops the loop

## Multiple statements in Initialization and Update Expressions

```
int x, y;  
for (x = 1, y = 1; x <= 5; x++) {  
    System.out.println(x + " plus " + y  
                        + " equals " + (x + y));  
}
```

1 plus 1 equals 2  
2 plus 1 equals 3  
3 plus 1 equals 4  
4 plus 1 equals 5  
5 plus 1 equals 6

```
int x, y;  
for (x = 1, y = 1; x <= 5; x++, y++) {  
    System.out.println(x + " plus " + y  
                        + " equals " + (x + y));  
}
```

1 plus 1 equals 2  
2 plus 2 equals 4  
3 plus 3 equals 6  
4 plus 4 equals 8  
5 plus 5 equals 10

# Nested Loop

- **Nested Loop:** a placing of one loop inside the body of another loop is called nesting.

```
for(initialization,; BooleanExpression; update){  
    for(initialization2,; BooleanExpression2; update2){  
        statement;  
        statement;  
        // Place as many statements here  
        // as necessary.  
    }  
}
```



# Nested Loop (Cont)

- **Example1:** webpage counter

43J048E1

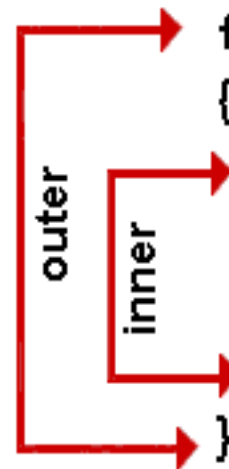
0	0
---	---

Considering 2 digits webpage counter...

```
for(num2 = 0; num2<=9; num2++)  
{  
    for(num1=0; num1<=9; num1++)  
    {  
        System.out.println(num2+ " "+ num1);  
    }  
}
```

outer

inner





# Nested Loop (Cont)

- **Example2:** Table position


```
for(int row=0; row<3; row++){  
    for(int col=0; col<3; col++){  
        System.out.println("table position=" + row + "," + col);  
    }  
}
```

# Nested Loop

## Practice 1...

```
for (int i = 1; i <= 4; i++) {  
    for (int j = 1; j <= 3; j++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```



# Nested Loop

## Practice 2...

```
for (int i = 1; i <= 4; i++) {  
    for (int j = 1; j <= i; j++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```



# do... while Loop

- do... while Loop: this is similar to while loop apart from that this loop is guaranteed to execute at least one time.

```
do{  
    statement;  
    // Place as many statements here  
    // as necessary.  
} while(booleanExpression);
```

# Do...while Loop

Practice1...

```
int i = 0;  
int sum = 9;  
do{  
    i++;  
    sum = sum + i;  
    System.out.println(i + ", " + sum);  
} while (sum < 10);
```



# Do...while Loop

- Practice2...
- To accept input from users, and keep asking until they input a negative number

```
Scanner scan = new Scanner(System.in);  
int val;  
do{  
    System.out.print("Please enter a negative number: ");  
    val = scan.nextInt();  
} while(val >= 0);  
  
System.out.println("Input value is " + val);
```