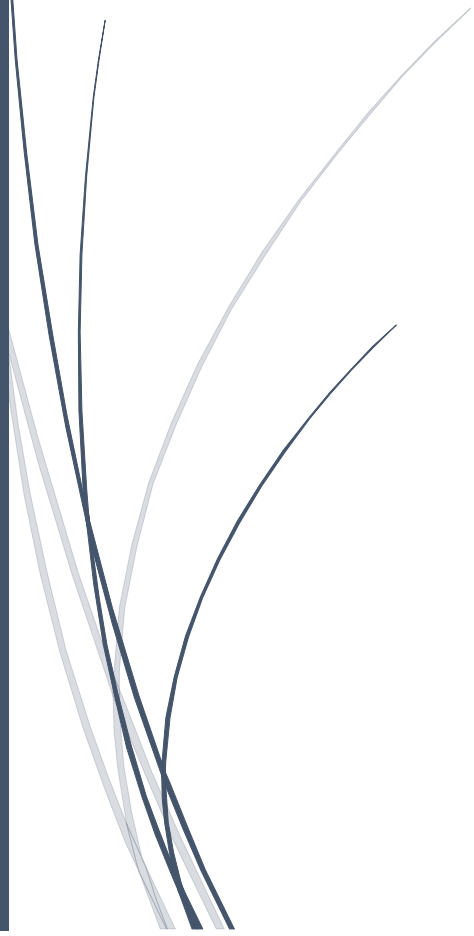
ACE Computer Science

ACS Java C

Student's Edition

A.C.E.



Academic Consulting & Education
WWW.ACECONSULT.ME

A.C.E.

Table of Contents

1. What Is Programming?

1) What can we do by programing?

We can build a castle, manufacture a car, and cook food. Similarly, we may make many different kinds of computer software by programing such as computer games, web sites, mobile apps, operating systems, and artificial intelligence programs. You may easily find computer software programs. You ride Uber, watch videos on Youtube, talk to your friends via WhatsApp, and find a friend on Facebook. All these are examples of using real computer software created by programing. We can help the world to be a better place by creating new programs that have never existed.

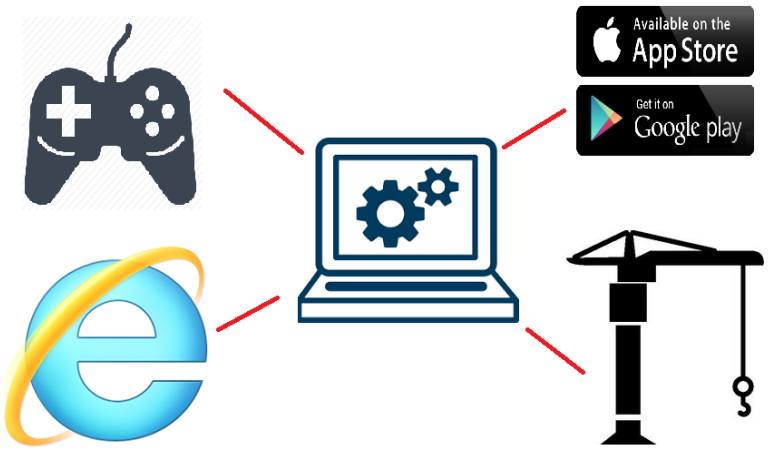


FIGURE 1 EXAMPLES OF APPLICATION PROGRAMS

Now, what is programing?

We saw people writing complex codes in science fiction films. They type so fast and look like geniuses. It makes you feel programing difficult. Only for geniuses. But it's not.

When we talk to a person, we speak in a language that he or she can understand. We call this language a "natural language." Imagine that you need to talk to computer. Unfortunately, computers can't understand the natural languages. Computers use a machine language called "assembly language." But assembly is impossible for a human being to understand because it is a series of 1's and 0's. So, computer scientists developed translator programs called compilers. We use a computer language which is easier for human and then compiler will translate the language to assambly.

FIGURE 2 PROGRAMMING LANGUAGE TO MACHINE LANGUAGE

There are many natural languages in reality such as English, Spanish, Chinese and so on. Just like this, many different programing languages have been developed such as C, C++, Java, Python, C# and so on. We may choose an appropriate language to solve a given problem. In this class, you are going to learn a programing language called "Java," which is the most popular programing language.

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	^	Python	4.678%	+1.21%
5	4	v	C#	3.754%	-0.29%

FIGURE 3 USAGE DISTRIBUTION OF THE COMPUTER PROGRAMMING LANGUAGES

Figure 3 shows the ranking of programing languages in year 2018. People use Java the most. Java is a productive and reliable language. We are going to study features of Java step by step.

We can easily guess that a Java complier is necessary to translate it to assambly. One of the compiler for Java, which we will use, is called "Eclipse." To use Java, let's install the development environment on your computer.

2) Install Java programming environment

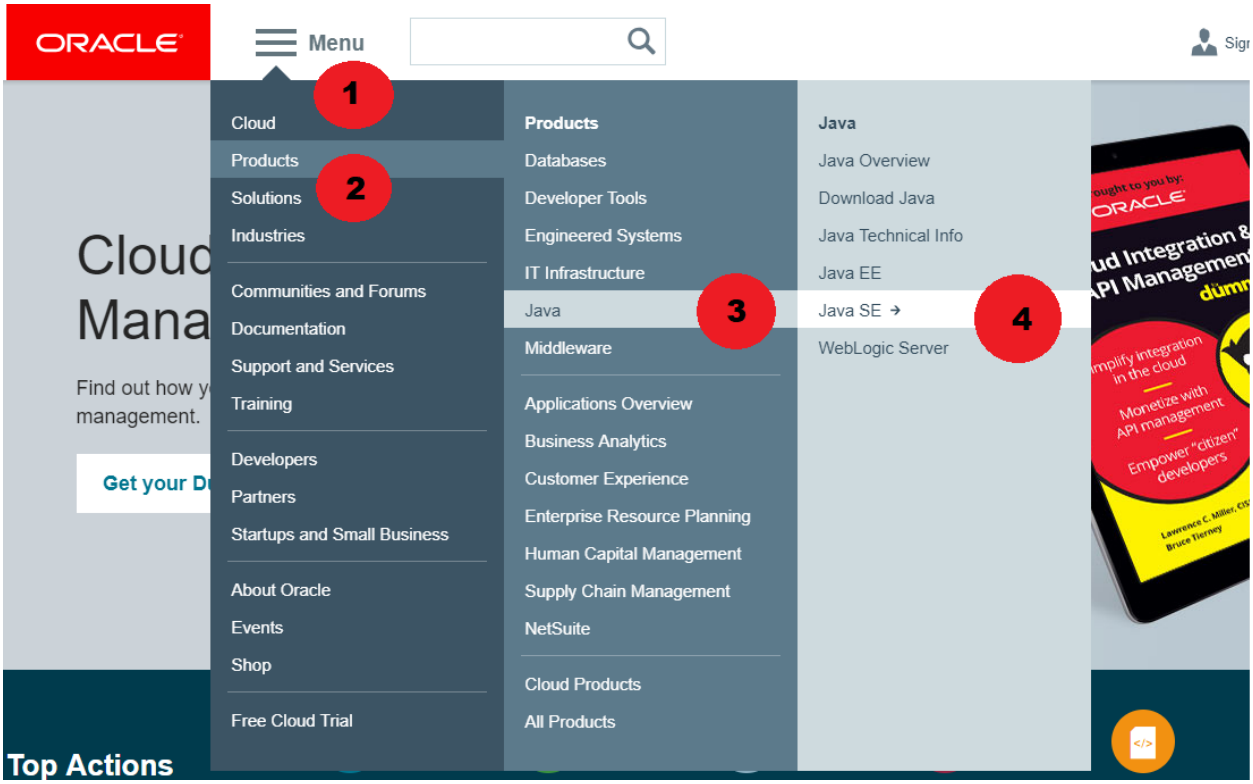
To use the Java language, we need to download and install two different programs.

1. JDK (Java development kit)
2. Eclipse (Java Compiler)

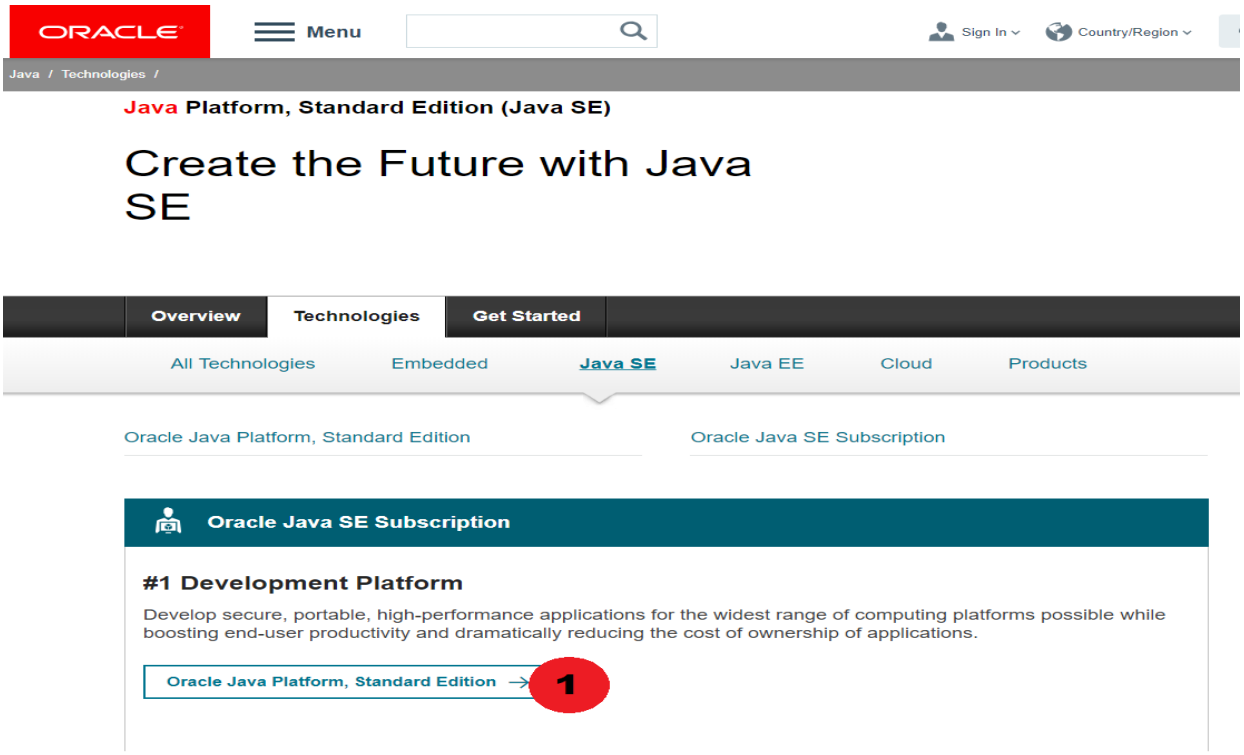
a) Install JDK

Step 1. Go to Oracle Web site https://www.oracle.com/index.html and click “Menu.”

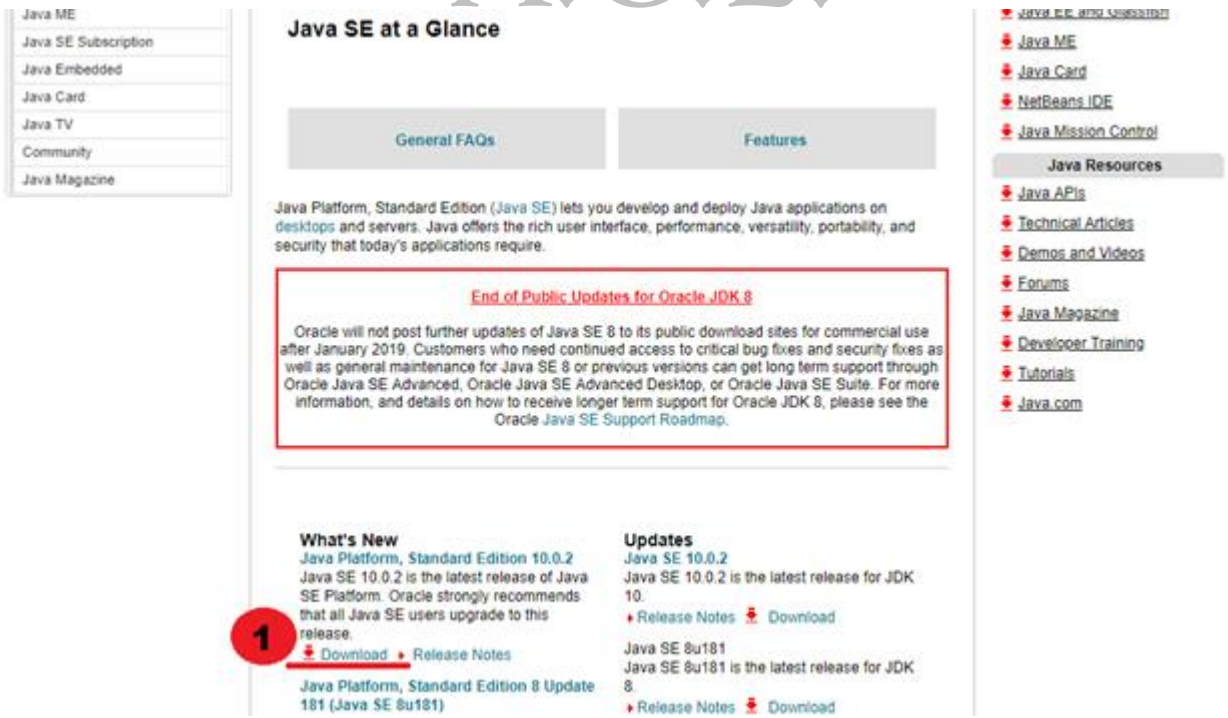
Step 2. Menu → Products → Java → Java SE



Step3. Click Oracle Java Pletform, Standard Edition



Step 4. The latest version is on the top. Download the new version.



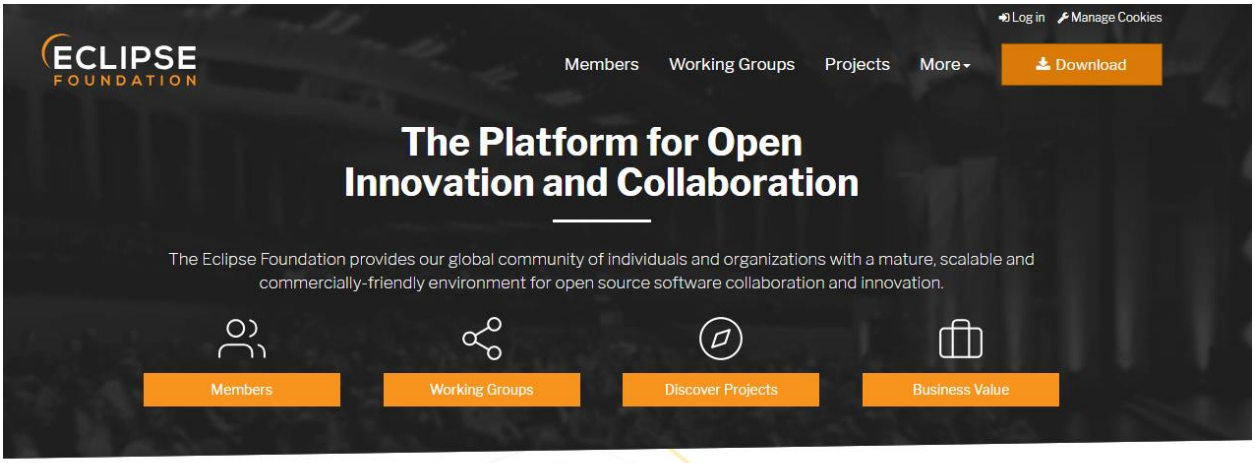
Step 5. Download JDK

Step 6. Select your OS and download the correct version. Mostly, macOS or Windows.

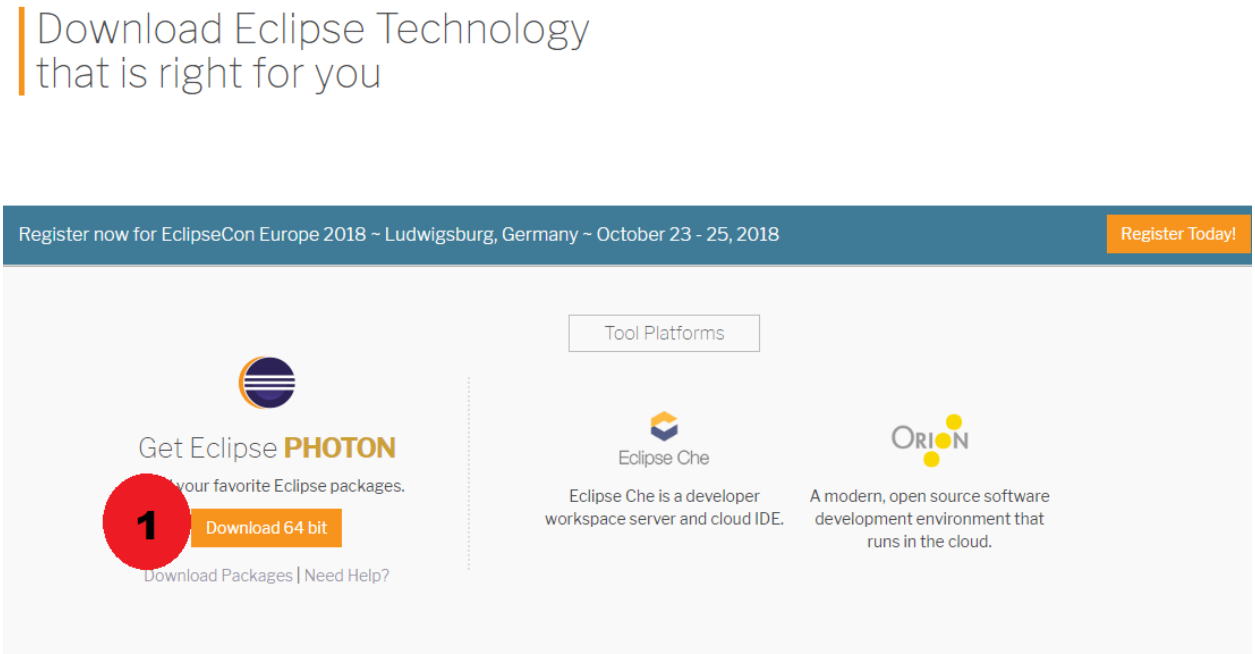
Step 7. Install the downloaded package.

b) Eclipse

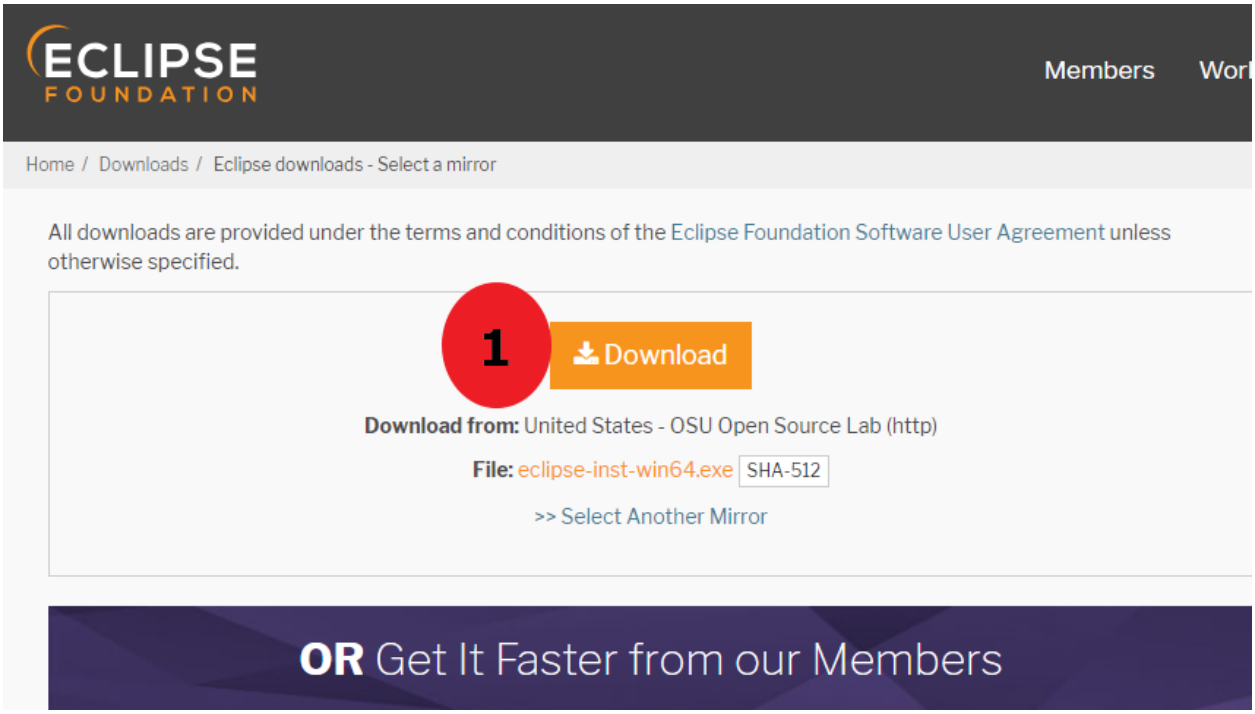
Step 1. Go to http://www.eclipse.org/ and click “Download” on the right top.



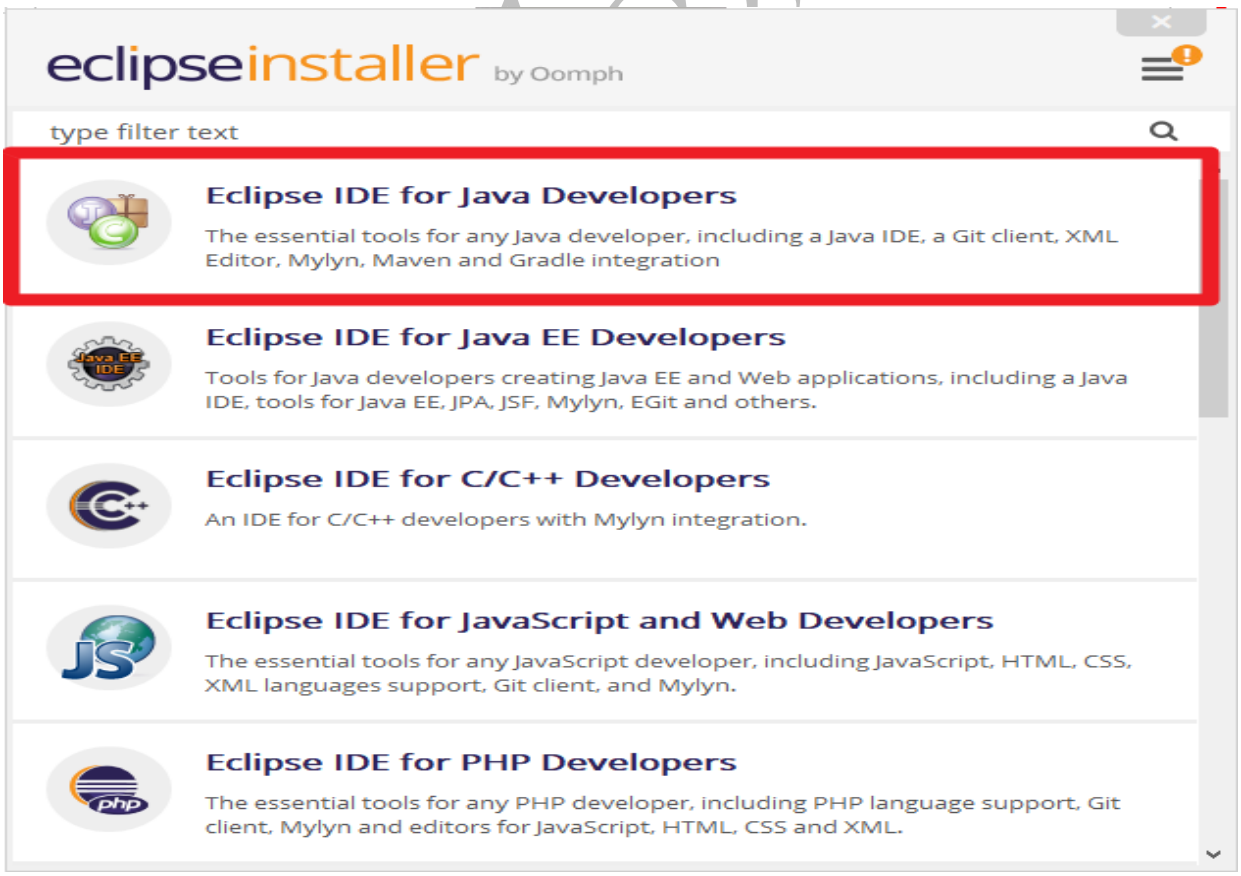
Step 2. Click download



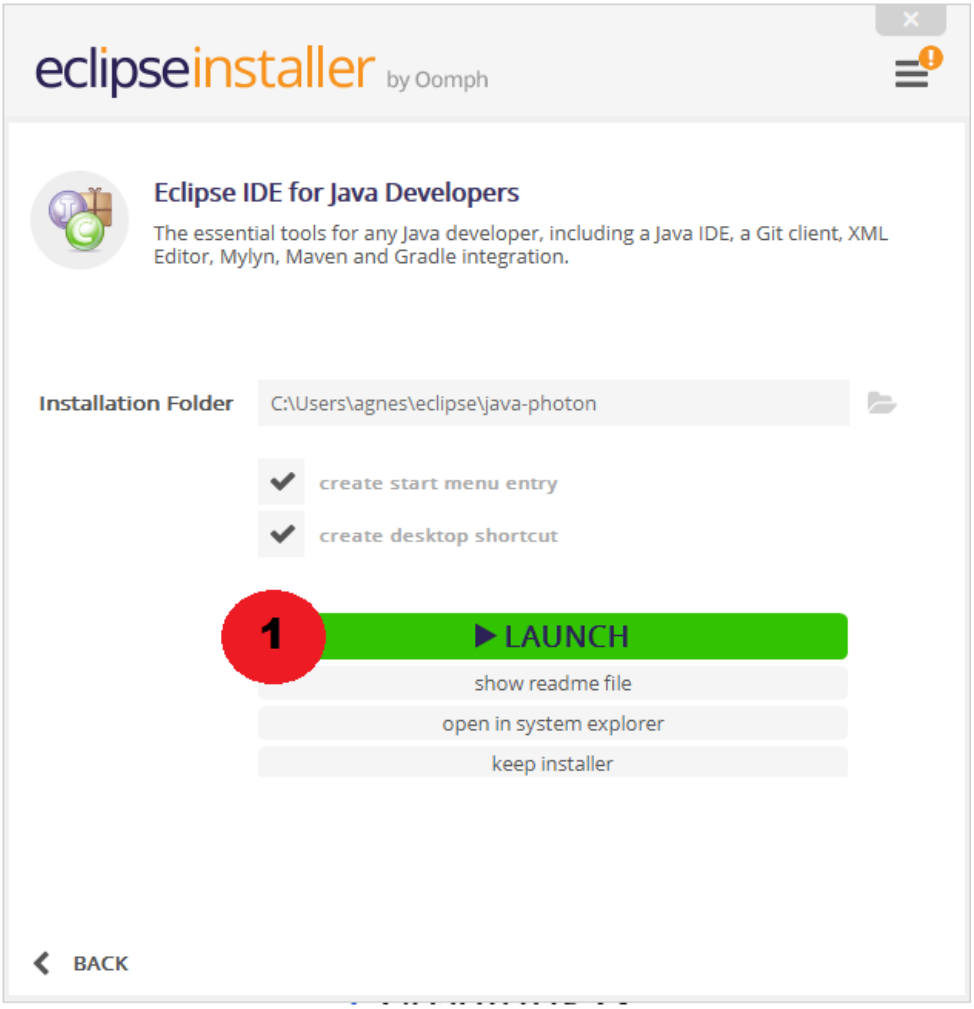
Step 3. Click download



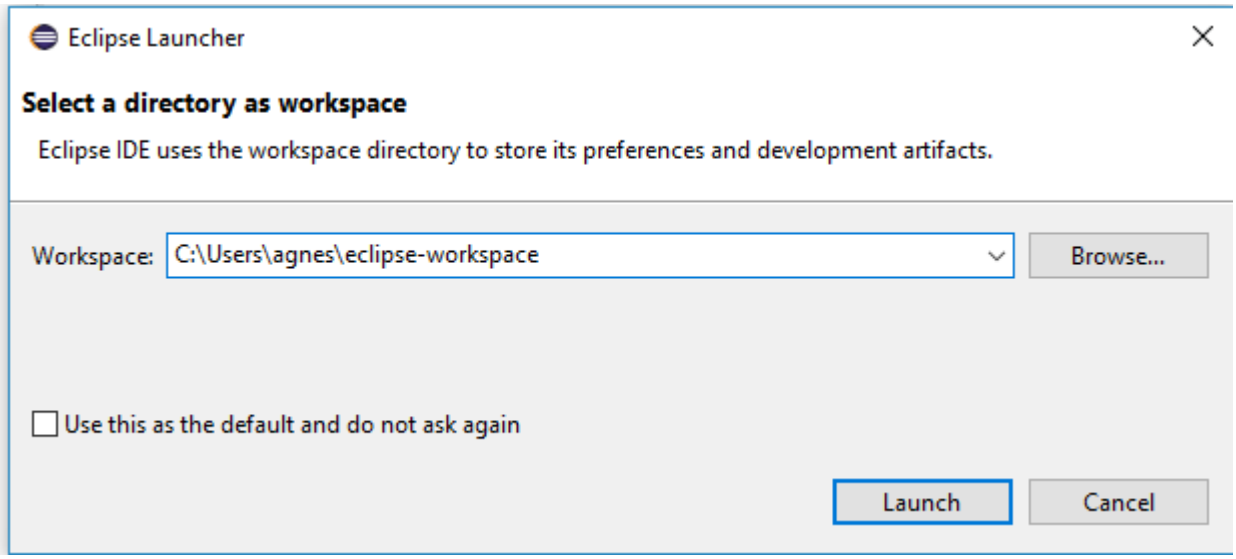
Step 4. Choose “Eclipse IDE for Java Developers” and click “Agree” to move forward.



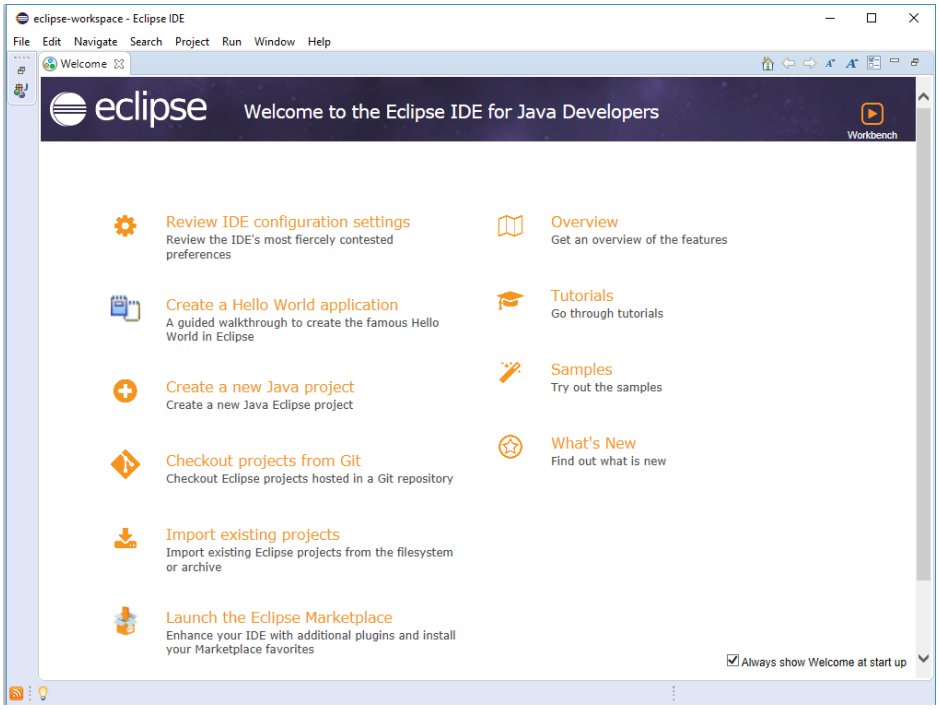
Step 5. After installation is complete, click “Launch.”



Step 6. Choose the path and click “Launch.” You may use the given path.

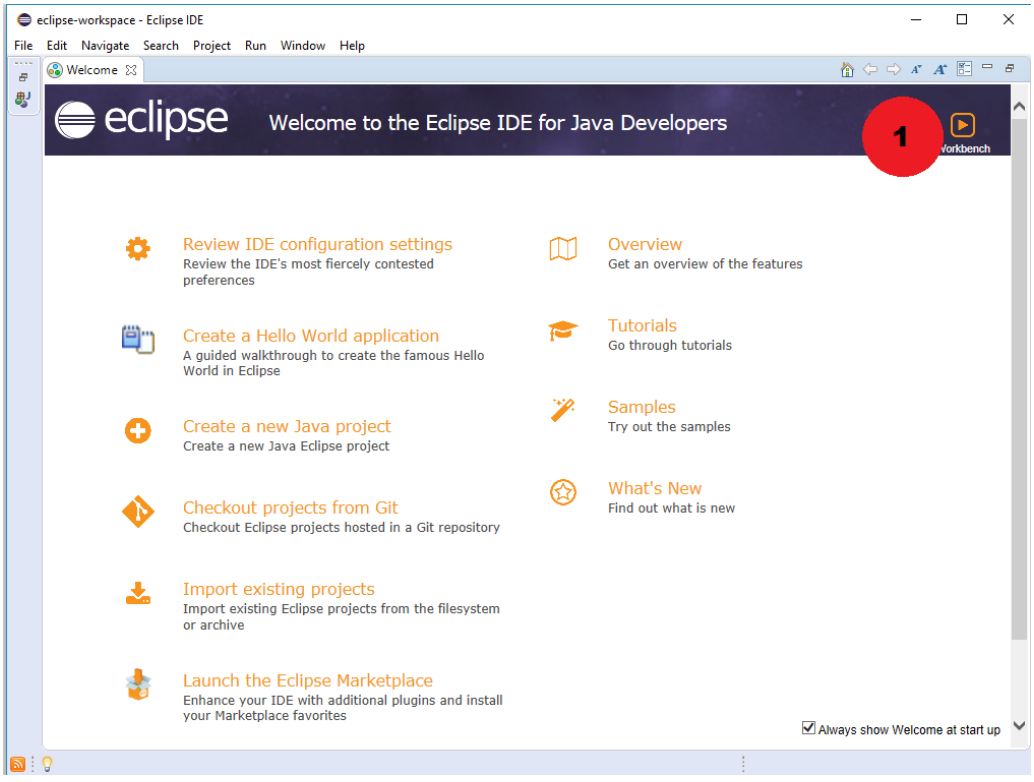


Step 7. If the nstallation was successful, you should see the below screen,

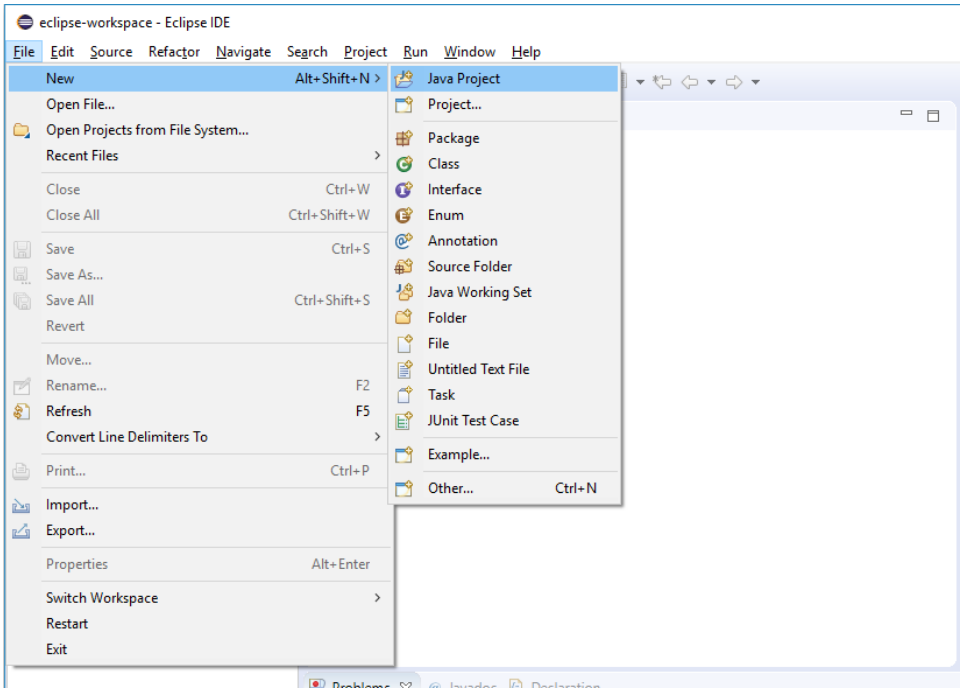


c) How to run Eclipse

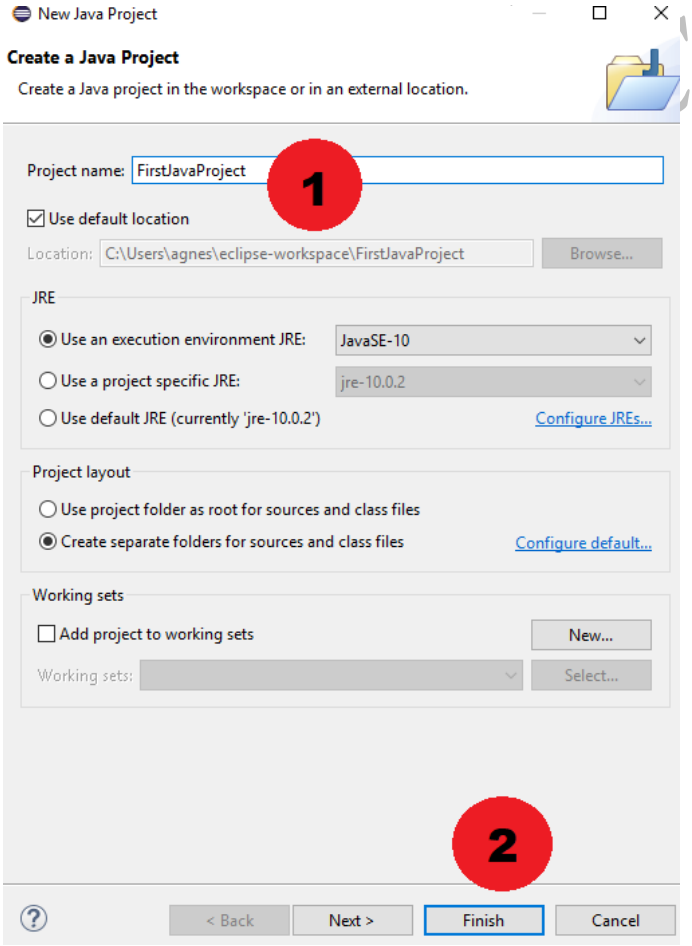
Step 1. Click “Workbench.”



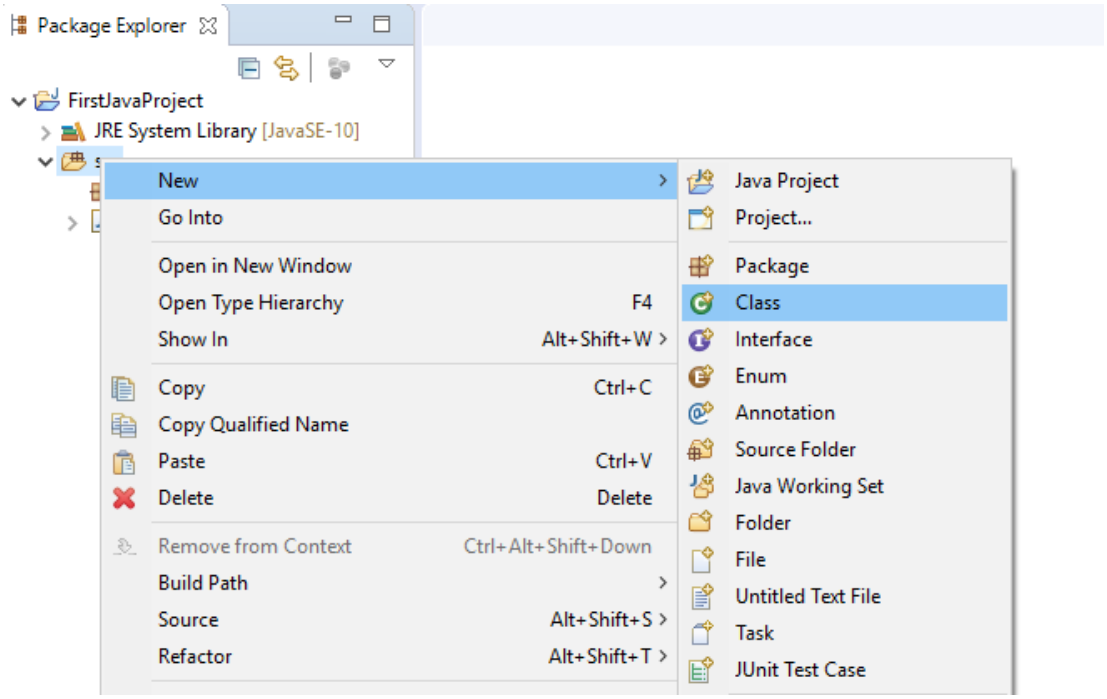
Step 2. File → new → Java Project



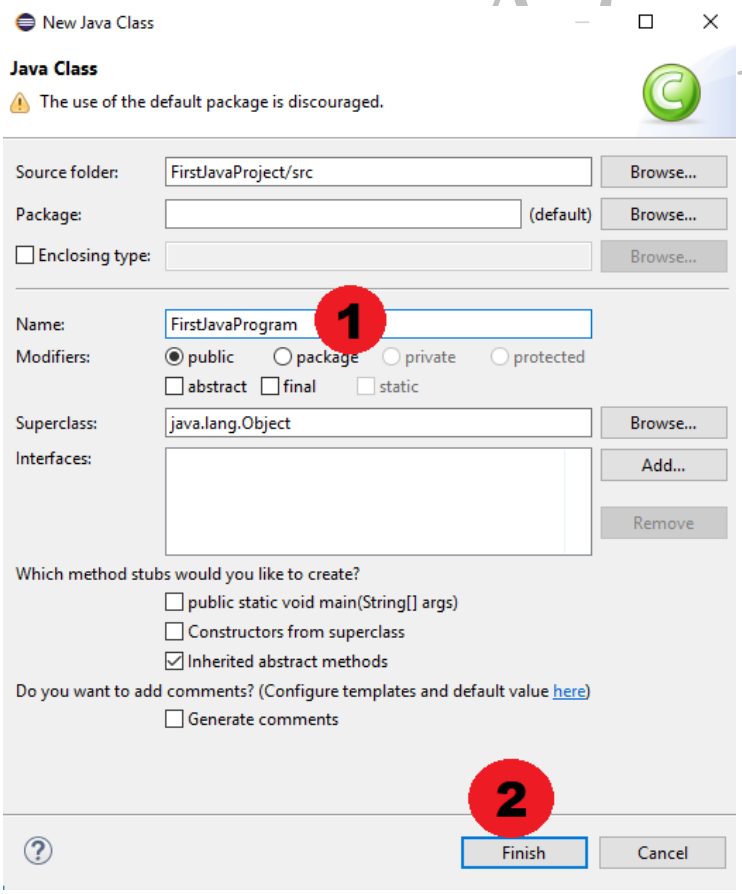
Step 3. Write “FirstJavaProject” on project name and click “Finish,” and then click “Create.”



Step 4. From Package Explorer toolbar on left, right click the folder called “src” and go to “new → class.”

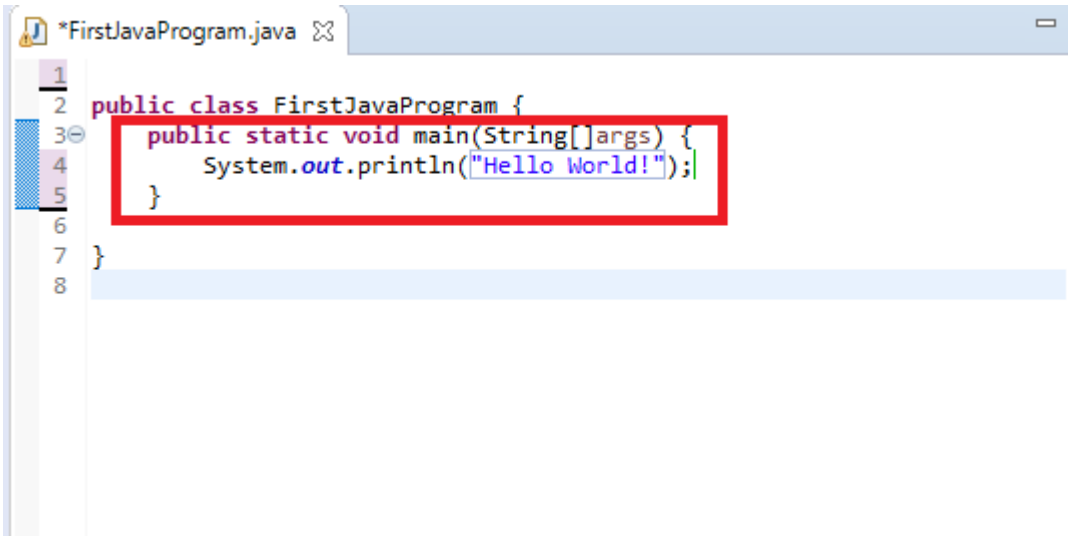


Step 5. Write “FirstJavaProgram” into “Name” box and click “Finish.”

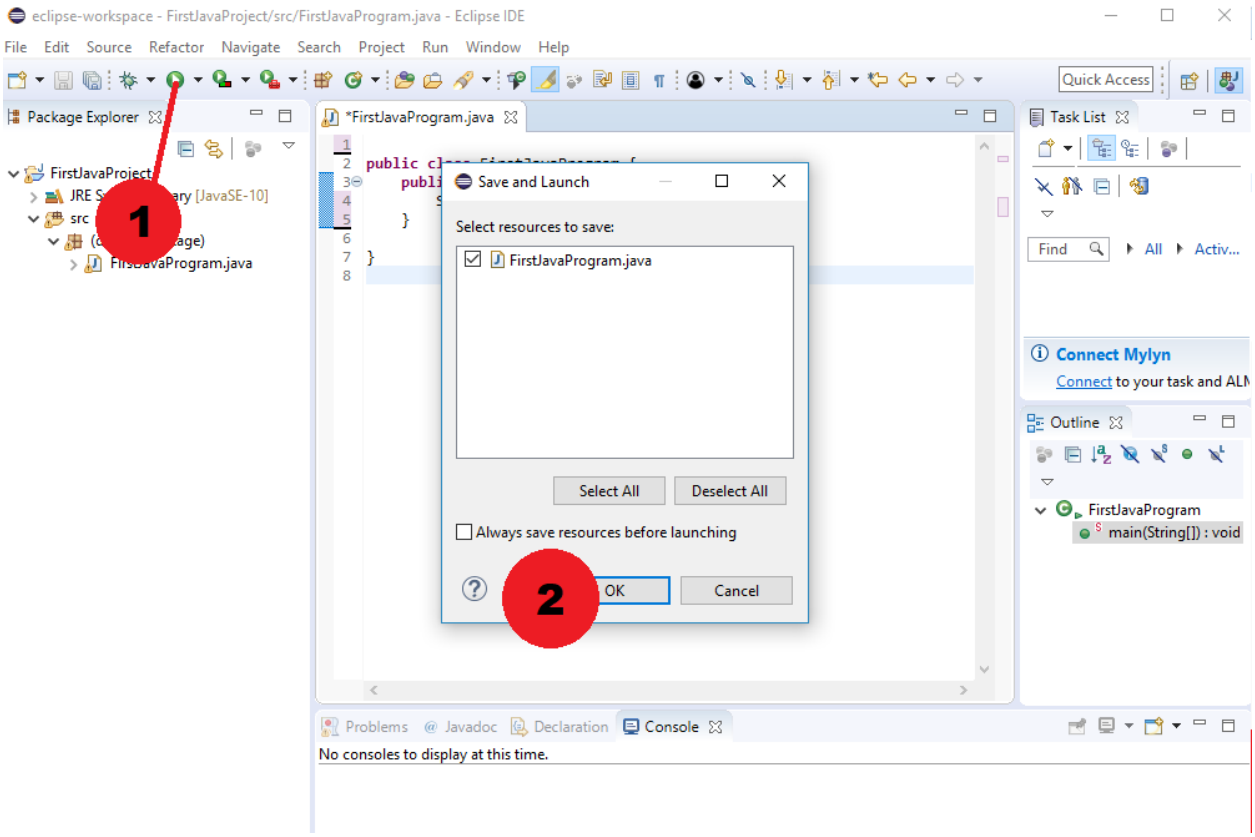


If there is a file named module-info.java, delete it.

Step 6. Write code exactly same as in the red box. (Note that you should be case sensitive.)



Step 7. Click “Run” button and click “OK” on the pop-up window.



Step 8. Do you see the result?

3) Output (println, print)

How do we talk with computer by using codes?

Let's think about how we talk each other. We make sounds by mouth and tongue. And there are rules in language. We call it grammar and words. The same concept works in programing too. We write code to talk. And there are pre-defined rules in programing languages.

Let's start with analyzing the first code while learning how to use Eclipse.

```
01  public class FirstJavaProgram {
02      public static void main(String[ ] args) {
03              System.out.println("Hello World!");
04      }
05  }
```

Just like reading a book, we start to read from the top.

a) The brackets

There are many brackets in a program. These brackets are just the same as bracket in math. For example let's look at $((5 * 3) + 2) * 2$. There are two pairs of brackets. They are always in pairs and surround the numbers. Brackets in a program express where the instruction starts and ends. And they make both computer and people understand codes easily.

If some codes start with a bracket, there will be extra space on the right. You can check line 02. Because the code is in a pair of brackets, there are more spaces in front. Using spaces to beautify the codes is called indentation by white characters. The indentation does not affect the functionality of the code.

b) The first line

Look at line 01:

```
public class FirstJavaProgram {
...
}
```

It shows that the name of your program is "FirstJavaProgram" and you are going to write your code inside the given brakets. We call it a class. We are not going to learn about class now, so let's just say it's kind of the name of your code. You don't have to care about this code because it will be automatically generated by Eclipse.

But you have to remember one thing. If the name of the class is different from the file name, it will cause an error. So if your file name is hello, the code should be "public class hello {}."

c) The second line ("main" method)

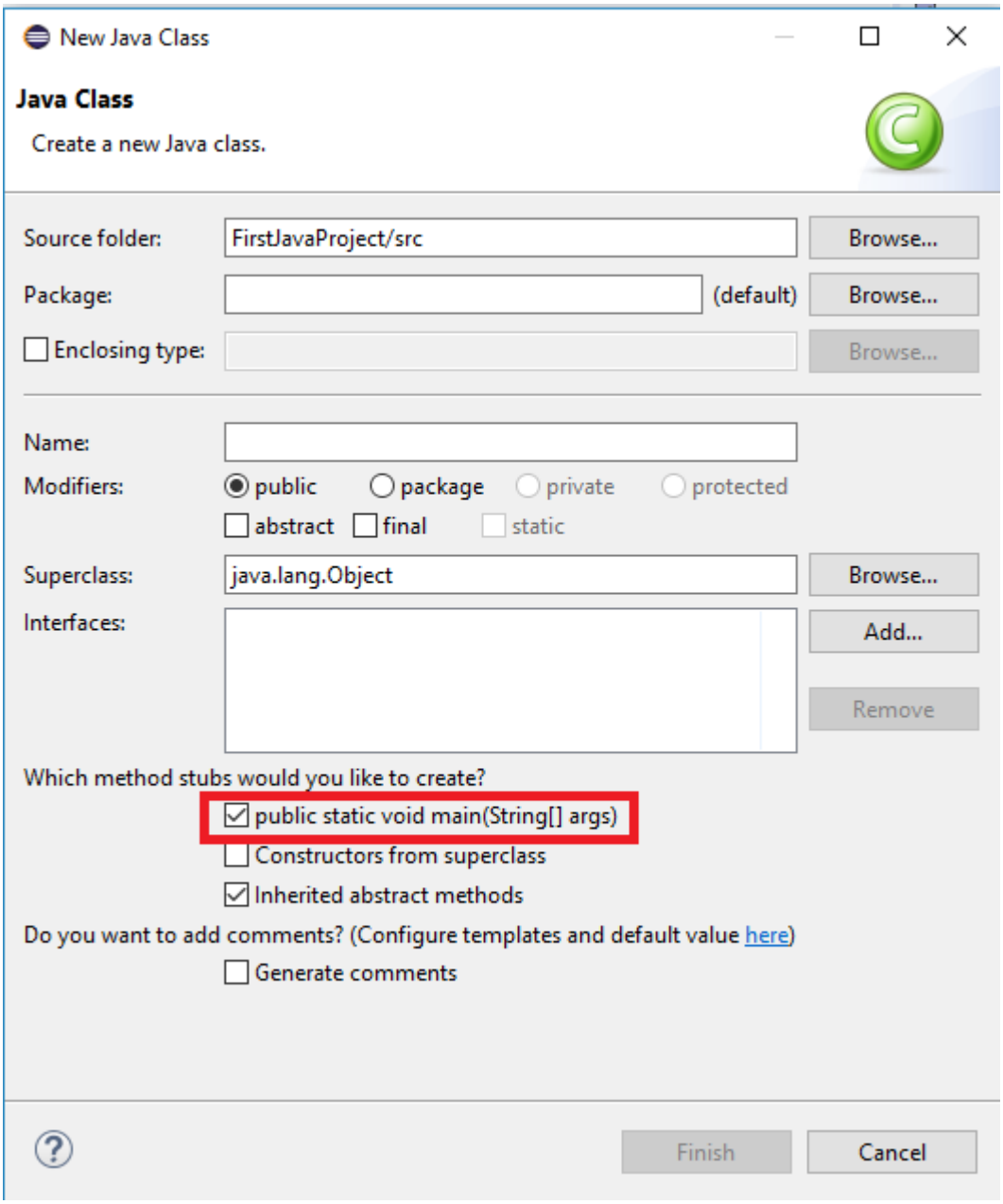
Second line is called main method. We are going to see this code very often because this code is essential for a program. When you run a program, even though there are many method in the program, the main method will be forced to be executed first. All your codes will be inside this main method.

```
    public static void main(String[ ] args) {
…
}
```

You will see these "main" method and "class" always, so even though you don't understand yet, just memorize that you have to include the following structure for a Java program.

```
public class ClassName {
        public static void main(String[ ] args) {
                //codes
                //codes
                //codes
                . . .
        }
}
```


In other words, "class" and "main" method is going to be the basic from of a Java program.



If you want Eclipse to generate the main method automatically, check the red-lined box, and the compiler will automatically include those codes. And you will write your codes inside the main method.

d) The third line

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

From the first project, it was the first code inside the main method. Now we are going to talk about the code you want to write. The code in the third line command the computer to print out everything in the bracket (or parentheses).

Standard format of the “println” statement is:

System.out.println("sentence");

where the underlined part is modifiable as you want and the meaning of println is to print out sentence and go to next line. Don't forget the semi-colon at the end of the line. The semi-colon directs the computer to recognize the end of a command.

Run FirstJavaProgram and see what happens.

e) The semi-colons

Computer does not recognize the end of a command or a line without a semi-colon. For example, if you write the code as follows:

```
1+1 = 2
2+2 = 4
```

A human knows that there are two commands (or codes). To the computer, it is same as 1+1=22+2=4. So, if you want to tell the computer that they are on separate lines, you have to add ";" at the end of each command. The correct codes should be:

```
1+1=2;
2+2=4;
```

Now, let's make a program which prints out:

```
Hello
World!
```


Example:

Output1.java

```
01 public class Output1 {
02          public static void main(String[ ] args) {
03                    System.out.println("Hello");
04                    System.out.println("World!");
05          }
06 }
```

Output:

```
Hello
World!
```

f) Another way to print

```
System.out.print("sentence");
```

Can you guess the difference from "println"?

Example:

Output2.java

```
01    public class Output2 {
02        public static void main(String[]args) {
03            System.out.print("Hello");
04            System.out.print("World!");
05        }
06    }
```

Note that the cursor location remain at the end of the line.

g) Printing two sentences using one println() statement

```
System.out.println("sentence 1" + "sentence 2");
System.out.println("sentence 1" + "sentence 2" " + "sentence 3" + …);
```

h) Types of printing

Sentence: Enclose the sentence with double quotation marks. Ex) "Harry Potter"

Numbers : just write the numbers. Ex) 324

Arithmetic equation : cover with bracket. Ex) (1+2).

Example:

Output3.java

```
01    public class Output3 {
02        public static void main(String[]args) {
03            System.out.println("Hello" + "World!");
04            System.out.println("I am " + 15 + " years old");
05            System.out.println("1 + 1 = " + (1+1));
06        }
07    }
```

Output

```
HelloWorld!
I am 15 years old
1 + 1 = 2
```

4) Homework

1. Let's make a program that prints "I'm going to be the ace in ACE."

2. Write a program that prints "Java is very easy." using four System.out.print() statements.

3. Write a program that prints "I can eat 5 hamburgers." by using four + operators.

4. Guess the output of the following Java program.

```
public class A_1_1 {
        public static void main(String[]args) {
                System.out.print(“Monday”);
                System.out.println(“Sunday”);
                System.out.print(“Friday”);
        }
}
```

Output:

5. Guess the output.

```
public class A_1_2 {
        public static void main(String[ ] args) {
                System.out.println(“I want it” + “that way”);
                System.out.println(“R” + “o” + “l” + “l” + “i” + “n” +”g” );
                System.out.println(“10 times 400 is ” + (400*10) + “.”);
        }
}
```

Output:

6. Find the error(s) in the following code.

```
public class A_1_3 {
        public static void main(String[]args) {
                System.out.println(“help” “me”);
                System.out.println(8 divided by 2 equal + (8/2) );
                System.out.println(“There might be no error”)
        }
}
```

Answer:

7. Construct a program to generate the following output.

Output:

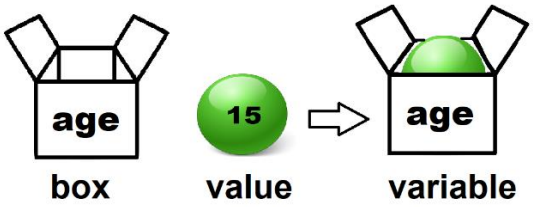
I don't want a lot for Christmas.
There is just one thing I need.

8. Write a program that prints out the answer of both equations.

Q1 : 4500 * 400 / 2 + 7132
Q2 : (432 + 35123) * (5122 + 9432)

2. Variables

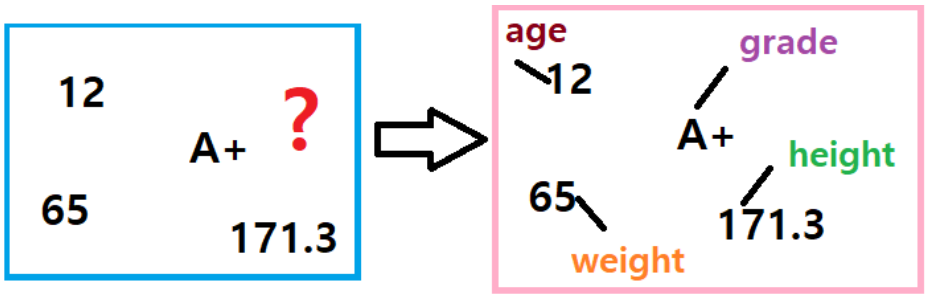
Variable means a number that can be changed. For example, your age will change next year, so we can call your age a variable. We don't call Pi a variable because the value of Pi does not change. It is called a **constant**.



There are two parts in a variable, which are box and value.

- Box: the name of the variable. The name of our variable is age.
- Value: the number associated with the variable.

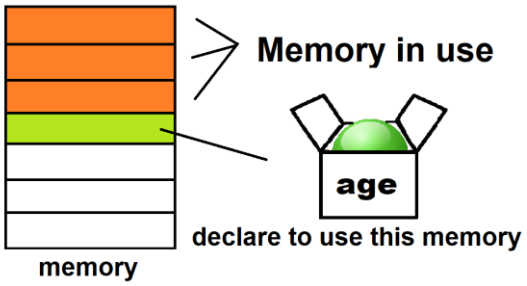
We store a value into a box. There can be many boxes in a program. When you want to know the value of age, find a box named age. You can change the value of the box later. Variables help you to organize many different types of data.



It would be very hard to recognize the meaning of each data if we don't consider the type of each data. Think of the data type as data tone. In music, each musical note represents different tone and length etc. For data types of a programming language, we need to define certain number of different tones as well.

1) Declaration of a variable

Do you know how big is the memory of your computer? 128GB? Or 1TB? 2TB? We are living in the world with TB memory. We know if we download something, the computer need memory to store them. For example: an mp3 file is about 5MB, a movie about 2~10GB. When you click to download something from the Internet, the computer will start downloading. In this process, you are getting permission from the computer to use some memories to store your file. To declare a variable is just same as that step. If you want to use a variable, you have to tell computer to prepare some memories for that variable so that other data won't make collision with that variable.



To declare a variable in a computer program, you will use the following format:

```
Type VariableName;
```

where Type defines the characteristics of the variable such as the size of memory.

2) Initialization of a variable

When you just declare a variable, the variable has no value. To make a variable have certain value, the variable has to be initialized..

```
VariableName = Value;
```

In the above sentence, = is the operator to assign a value to the declared variable. This = operator works differently from mathematical equal sign.

Example: The following program declares a variable named age, and the variable contains the information of your age.

Type1.java

```
01  public class Type1 {
02      public static void main(String[ ] args) {
03          int age;
04          age = 15;
05
06          System.out.println("I am " + age + " years old.");
07      }
08  }
```


Output:

```
I am 15 years old.
```

Java has many different types and one of them is “int,” which means integer. We will study further about this in “e) Data types in Java.”

3) The = operator in Java

- A = 10; assign 10 to variable A
- A = B; fetch the value of variable B and assign that value to variable A
- B = 2 + 3; Calculate 2 + 3 first and then assign the result to B

So the three essential parts of a variable are **Type**, **VariableName**, and **value**. In Java, we need to declare / initialize all variables before they are used.

4) Declaring and initializing variables

Type VariableName = value;

Example:

int age = 15;

Type VariableName1, VariableName2, …;

Example:

int age, height;
int age, height, weight, grade;

Type VariableName1 = value1, VariableName2 = value2, …;

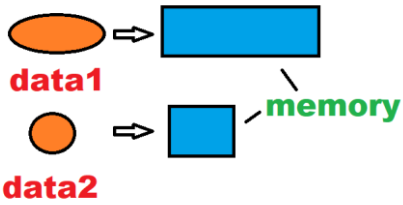
Example:

int age = 15, height = 170;
int age = 15, height = 170, weight = 65, grade = 9;

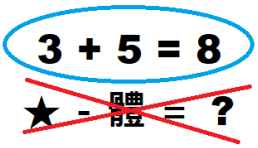
5) Data types in Java

The concept of values and variable names are clear. But still data types may be confusing. What is "int"? Data types mean specific formats of data. There are a number of types in Java.

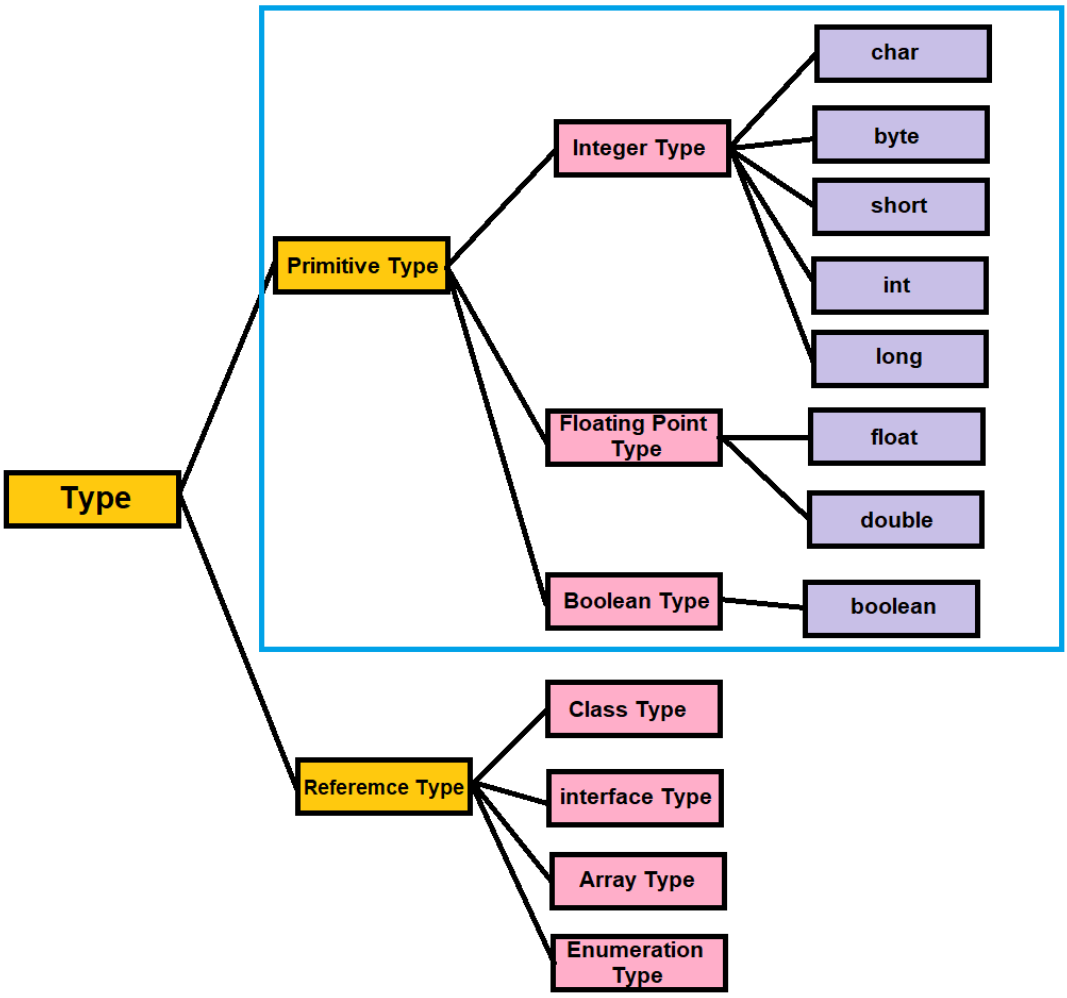
Let's assume that the value of data1 is "Hello my name is Aaron." And the value of data2 is 5. Will these two variables take same amounts of memory? We can easily guess that data1 needs more space than data2. We need to take different amount of memory when we declare different variables for not wasting memory. So, we decide the size of memory for each type and get only that much memory for the variable.



Will a computer recognize pi + 200? Probably not. A word can't be increased as numbers. Computers need to know if the data can be calculated or not. Only same type of operands can be calculated.



Thes following diagram shows the data types pre-defined in the Java language.



We are only going to learn about the types in the box. Primitive types are offered for efficient execution: **integer**, **floating point**, and **Boolean** types. They express numbers in different ways.

Type	Bits	Range	example
byte	8	-128 ~ 127	1
short	16	-32,768 ~ 32,767	2
int	32	-2,147,483,648 ~ 2,147,483,647	8
long	64	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	15
char	16	0 ~ 65,535	'a'
float	32	1.4E – 45 ~ 3.4028235E38	3.14f
double	64	4.9E-324 ~ 1.7976931348623157E308	3.14d
boolean	8	0 ~ 1	True or 0

In the chart, bits shows the size of data type. Byte and Boolean has the shortest size of memory whilst long and double has the longest size of memory. It decides the range of the numbers that each type can represent.

6) Integer types (byte, short, int, long)

The difference between byte, short, int, long is how big a number each type can hold. Note that integer types can represent only integer numbers, not fraction or decimal.

Let's say there is a variable which shows a person's age. It may not be more than 120. It cannot be like 2382398928323 years old. To save memory, we prefer to use byte type instead of using long type.

To represent a letter, char type is used. It is one of integer types. Because each letter (character) has its own number. We call this mapping ASCII system.

DEC	Char	DEC	Char	DEC	Char	DEC	Char	DEC	Char	DEC	Char
0	Ctrl-@ NUL	22	Ctrl-V SYN	43	+	65	A	86	V	108	l
1	Ctrl-A SOH	23	Ctrl-W ETB	44	,	66	B	87	W	109	m
2	Ctrl-B STX	24	Ctrl-X CAN	45	-	67	C	88	X	110	n
3	Ctrl-C ETX	25	Ctrl-Y EM	46	.	68	D	89	Y	111	o
4	Ctrl-D EOT	26	Ctrl-Z SUB	47	/	69	E	90	Z	112	p
5	Ctrl-E ENQ	27	Ctrl-[ESC	48	0	70	F	91	[	113	q
6	Ctrl-F ACK	28	Ctrl-₩ FS	49	1	71	G	92	₩	114	r
7	Ctrl-G BEL	29	Ctrl-] GS	50	2	72	H	93	]	115	s
8	Ctrl-H BS	30	Ctrl-^ RS	51	3	73	I	94	^	116	t
9	Ctrl-I HT	31	Ctrl_ US	52	4	74	J	95	_	117	u
10	Ctrl-J LF	32	Space	53	5	75	K	96	`	118	v
11	Ctrl-K VT	33	!	54	6	76	L	97	a	119	w
12	Ctrl-L FF	34	"	55	7	77	M	98	b	120	x
13	Ctrl-M CR	35	#	56	8	78	N	99	c	121	y
14	Ctrl-N SO	36	$	57	9	79	O	100	d	122	z
15	Ctrl-O SI	37	%	58	:	80	P	101	e	123	{
16	Ctrl-P DLE	38	&	59	;	81	Q	102	f	124	|
17	Ctrl-Q DC1	39	'	60	<	82	R	103	g	125	}
18	Ctrl-R DC2	40	(	61	=	83	S	104	h	126	~
19	Ctrl-S DC3	41	)	62	>	84	T	105	i	127	DEL
20	Ctrl-T DC4	42	*	63	?	85	U	106	j		
21	Ctrl-U NAK			64	@			107	k		

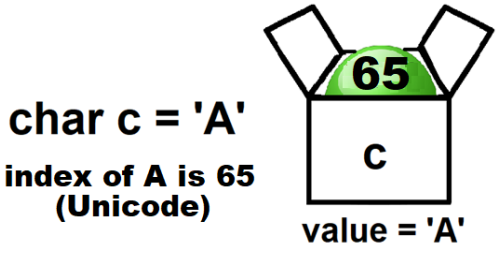
Table of ASCII system

7) Character type (char)

In the Unicode table, Dec means decimal number of each character. For an index 65, the corresponding character is A. There are 1,048,576 characters in the world. And Unicode covers these many characters. The table above shows just a part of Unicode characters.

In the diagram at the right, the variable 'c' is declared as char type and initialized to have the value of 'A'.

And the index of 'A' in Unicode is 65. Computer keeps only the index of 'A', so the box(c) contains 65 and when the computer generates the output of c, it find the character which matches the index from the Unicode table.



Only one letter can be represented with char type. When you initialize a char type variable with a character, the character should be embraced with single quotation marks as in the code below.

Example:

CharType.java

```
01      public class CharType {
02              public static void main(String[ ] args) {
03                      char c1 = 'A';
04                      char c2 = 65;
05
06                      System.out.println(c1);
07                      System.out.println(c2);
08              }
09      }
```

Output

```
A
A
```

8) Special characters

Special characters	Designation method	Unicode value in hex
Backspace	\b	\u0008
Tab	\f	\u0009
Linefeed	\n	\000A
Formfeed	\f	\u000C
Backslash	\\	\u005C
Single Quote	\’	\u0027
Double Quote	\”	\u0022

Example:

```
char c1 = ‘\t’;          // c1 holds tab character
char c2 = ‘\n’;          // c2 holds linefeed character
```

9) Floating point types (float, double)

Floating point types can hold decimal numbers. Double type can represent bigger numbers than float types and can save more accurate numbers. On the other hand double type uses more memeory than float type.

The default type of a floating point number is double. So, if you write a floating point constant like 3.14, it will automatically become double type.

Write 'f' after a decimal number for float type, and append 'd' or nothing for double type.

Example:

FloatAndDouble.java

```
01  public class FloatAndDouble {
02      public static void main(String[ ] args) {
03          float  a = 12345678901234567890.0f;
04          double b = 12345678901234567890.0;
05
06          System.out.println("float variable a's value : " + a);
07          System.out.println("double variable b's value : " + b);
08
09          float c= 1.0f/3.0f;
10          double d = 1.0/3.0;
11
12          System.out.println("float variable c's value : " + c);
13          System.out.println("double variable d's value : " + d);
14      }
15  }
```

Output:

```
float variable a's value : 1.2345679E19
double variable b's value : 1.2345678901234567E19
float variable c's value : 0.33333334
double variable d's value : 0.3333333333333333
```

From the above, 1.2345679E19 means 1.2345679×10^{19}

Also, note that float type prints up to 8^{th} digit of a decimal number and double up to 16^{th} digit.

10) Boolean type

Boolean type only has two values: true or false.

Example: BooleanVariables.java

```
01    public class FirstJavaProgram {
02        public static void main(String[ ] args) {
03            boolean a = true;
04            System.out.println("The value of boolean variable a is " + a + ".");
05
06            boolean b = 10 > 20;
07            System.out.println("The value of boolean variable b is " + b + ".");
08
09            boolean c = a;
10            System.out.println("the value of boolean variable c is " + c + ".");
11        }
12    }
```

Output:

```
The value of boolean variable a is true.
The value of boolean variable b is false.
The value of boolean variable c is true.
```

11) String

Do you still remember your first program? The first program had only one sentence:

System.out.println(“Hello world!”);

We learned many kinds of variables. But we still didn’t have a variable that can hold “Hello world!”. Maybe char type? Only one letter can be saved with char type. How can we save a whole sentence in one variable?

In this case, we use String type. Be careful that the first letter S should be written as a capital.

The syntax to declare and initialize a String is:

String variableName = “AnyString”;

You have to write AnySentence inside double quotation marks. String can contain spaces and special letters as well.

Example: StringSample.java

```
01    public class StringSample {
02        public static void main(String[ ] args) {
03            String a = "Hello ";
04            String b = "World!";
05
06            int i1 =1000;
07            int i2 =2000;
08
09            System.out.println(a + b);
10            System.out.println(a + "World!");
11            System.out.println(a + i1 + i2 + b);
12            System.out.println(i1 + i2);
13        }
14    }
```


Output:

```
Hello World!
Hello World!
Hello 10002000World!
3000
```

String is a complicated type. There are lots of methods in this type. We are going to learn useful methods of String later.

12) Type casting

We have learned many data types in Java. Some types are similar. Can we change a type to a similar type? The answer is yes.

The syntax of forced type cast is:

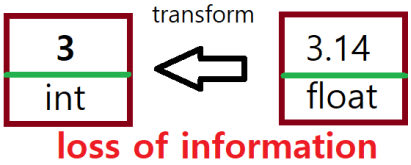
(type) variableName
(type) number

This is not always possible. And if you try to cast one type to another, there might be loss of data. To understand this, there is the priority among types.

byte >> short/char >> int >> long >>float >> double

Left type is has less priority than right type, which means that you may cast left type to right type without loss of data. To understand this, look at the figure below.

You have a float type variable with the value of 3.14. What happens if you cast it to int type? The int type can't have decimal part. So the decimal part in a float type variable will not fit the int type variable. There need to be some adjustment, which should be to round off the decimal part.



So the value will be changed to 3. There is data loss. Casting left to right is called widening, and casting right to left is called narrowing. When we are narrowing, we need forced cast, but for widening, we don't.

Example:

```
01  public class example {
02      public static void main(String[ ] args) {
04          byte   b = 120;
05          int    i= b;
06          System.out.println("Widening : " + i);
07
08          int    j = 259;
09          b = (byte) j;
10          System.out.println("Narrowing : " + b);
11
12          double d = 259.428;
13          i = (int) d;
14          System.out.println("double 259.428 to int : " + i);
15      }
16  }
```

Output:

```
Widening : 120
Narrowing : 3
double 259.428 to int : 259
```

There is one thing you may not understand. What do you think is the reason of second line of output?

Let's go back to the chart on page 28. The range of a byte variable is $-128 \sim 127$. But the value of variable j is 259. What will happen?

This number was not what we expected. We call this Situation "Overflow." For now, let's just note easily that the result is unpredictable when overflow happened. Overflow is one of common errors the programmers would make because it could be made during the mathematical calculation.

Conversion between String and int

String can contain any Unicode character. Imagine that you have a string which looks like an integer, it is different from int type number. That string consists of numeric letters only. So the string cannot be used for arithmetic operations.

String type can't change with forced casting. There are special ways to cast. Let's start with casting between int and String

The syntax of converting String to int, or vice versa is:

```
Integer.parseInt(String)
Integer.toString(int)
```

First line is used to convert from String to int. Second line from int to String.

What if you want to add two numbers in String type? Ex) String a("130") and String b("382") Because the type of these two numbers are String, if you do a + b, you will get "130382"

To get $130 + 382 = 512$, You have to cast String to int.

Example: StringToInt.java

```
01    public class StringToInt{
02        public static void main(String[ ] args) {
03            String a1 = "130";
04            String b1 = "382";
05            System.out.println("a+b = " + (a1+b1));
06
07            int a2 = Integer.parseInt(a1);
08            int b2 = Integer.parseInt(b1);
09            System.out.println("(int)a+(int)b = " + (a2+b2));
10        }
11    }
```

Output:

```
a+b = 130382
(int)a+(int)b = 512
```

13) Formatted printing

Sometimes, we just need to use two decimal places. Use the following sentence.

System.out.format("FormatString", variable1, variable2, …);

If you use "%.5f" for the FormatString, you can print up to 5th decimal place. Similarly, with %.3f, you can print up to 3rd decimal place.

Example:

FloatingPoints.java

```
01  public class FloatingPoints {
02      public static void main(String[] args) {
03          float f= 1.0f/3.0f;
04          double d = 1.0/3.0;
05  // Note that the special character \n is used for linefeed.
06          System.out.format("%.5f \n", f);
07          System.out.format("%.3f \n", d);
08          System.out.format
09              ("Multiple numbers in a line like %.4f and %.7f", f ,d);
10      }
11  }
```


Output:

```
0.33333
0.333
Multiple numbers in a line like 0.3333 and 0.3333333
```

In line 9, there are two variables in a line. You can put multiple variables but the number of % has to be the same as the number of variables.

14) Homework

1. Figure out if each identifier is acceptable or not.

   String first name = “John Doe”; ( )
   int child-age = 10; ( )
   double _distance = 35.92; ( )

2. Write a program that stores your height and print it.

3. Declare three variables for your age, the number of your siblings, and how many days you studied Java. Use the variables to print out each value with simple description.

4. Write a program which has three char type variables and each of them are written in ASCII index. Each char variable has 65, 67, and 69, respectively. Print all three variables in a single line.

5. Build a program that has a variable containing “999999999999999.0” and print this number without any loss.

Output

6. Assume that π is 3.14159265359. If the radius of a circle is 5, find the circumference and print it out to the second decimal place.

Output:

7. You have three Boolean variables: “juniorHighSchoolStudent”, “moreThanFiveFriends”, “ateDinner”. Each value describes you. Print each value as in sample..

Sample output:

```
I'm a junior high school student : true
I have more than five friends : false
I ate dinner : false
```


8. Predict the output of the following program.

```
public class example {
        public static void main(String[] args) {
                int days = 30;
                String subject = "math";
                float score = 89.5f, scoreForA = 90.0f;
                char grade = 'A';

                System.out.println("I studied "+subject+" for "+days+" days, and I got
"+score+" on this test.");
                System.out.println("It was so close that I needed "+(scoreForA-score)+" more
points to get " + grade + “.”);
        }
}
```

Output:

9. There are two int variables and two String variables. Use them to print below.

Output:

300 times 400 is 120000.

10. Assume that π is 3.14159265359. Radius of a circle is 5. You are going to calculate the area of the circle and put it in a double type variable. You don't want to print decimals. (convert to int)

Output:

78

11. There are two int type variables. The value of each variable is 100 and 200, respectively. You are going to print String “100200” using these variables.

12. Predict the output of the following program.

```
public class example {
        public static void main(String[] args) {
                float f = 100.32f;
                int a;
                char c;
                a = (int)f;
                c = (char)a;
                f = a;

                System.out.println(a);
                System.out.println(c);
                System.out.println(f);
        }
}
```


Output:

13. Predict the output of the following program.

```
public class example {
        public static void main(String[] args) {
                byte b = 100;
                int i;
                i = b;
                System.out.println(i);

                i = 300;
                b = (byte) i;
                System.out.println(b);
        }
}
```

Output:

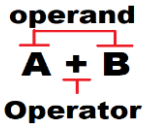
14. Find error(s) if any.


```
public class result {
        public static void main(String[] args) {
                byte b;
                int i = 300;
                long j = 9876543210;
                b=i;
                i=j;
                System.out.println(b);
                Sytem.out.println(i);
                Sytem.out.println(j);
        }
}
```

A.C.E.

3. Operators and Input

How many operators do you know? +, −, ∗, / and more? There are more operators in Java. Mostly, an operator has two operands as in the figure below.



1) The number of operands

We have three different kinds of operators in terms of the number of operands.

- Unary operator : only one operand
  Ex) a++
- Binary operator : two operands
  Ex) 1 + 1
- Ternary operator : three operands
  Ex) a > 3 ? true : false

2) Types of operators

a) Arithmetic operators

Four basic arithmetic operators are addition, subtraction, multiplication, and division. There are many ways for arithmetic expressions to construct efficient codes.

Arithmetic operators in Java

operator	Way to use	explanation	remarks
+	op1 + op2		Unary/Binary
−	op1 - op2		Unary/Binary
*	op1 * op2		Binary
/	op1 / op2		Binary
%	op1 % op2	The remainder of op1 when divided by op2	Binary
+ +	var++	Evaluate var before var is increased by 1.	Unary
	++var	Evaluate var after var is increased by1.	Unary
− −	var--	Evaluate var before var is decreased by 1.	Unary
	--var	Evaluate var after var is decreased by 1.	Unary

Note that ++ and -- are only for variables, not for constants.

Example: Operators.java

```
01  public class Operators {
02      public static void main(String[] args) {
03          int a=6, b=4;
04          int sum = a+b;
05          System.out.println("a+b=" + sum);
06
07          int sub= a - b;
08          System.out.println("a-b=" + sub);
09
10          int mul = a*b;
11          System.out.println("a*b=" + mul);
12
13          int div = a/b;
14          System.out.println("a/b=" + div);
15
16          int c = ++a;
17          System.out.println("(pre-increase)c=" + c);
18          System.out.println("(pre-increase)a=" + a);
19
20          int d = b++;
21          System.out.println("(post-increase)d=" + d);
22          System.out.println("(post-increase)b=" + b);
23      }
24  }
```


Output:

```
a+b=10
a-b=2
a*b=24
a/b=1
(pre-increase)c=7
(pre-increase)a=7
(post-increase)d=4
(post-increase)b=5
```

Note lines 16 and 20. ++ means increasing the value of operand by one, but the timing of increment is different.

- a = ++b Add 1 to b first, then put the value of b into a.
- a = b++ Put the value of b into a, then, increase the value of b by 1.

These operators seem a little confusing. If you use this operator in right way, however, it will be very useful.

b) Relational operators

These operators compare operands and find out the relation between the operands. There are only two ways to tell the relation: true or false. If those operands have different types, it will automatically change the type which has less priority to the higher one.

Relational operator

Operator	Way to use	Description
>	op1 > op2	If op1 is bigger than op2
>=	op1 >= op2	If op1 is bigger than or equal to op2
<	op1 < op2	If op1 is smaller than op2
<=	op1 <= op2	If op1 is smaller than or equal to op2
==	op1 == op2	If op1 is equal to op2
!=	op1 != op2	If op1 is not equal to op2
instanceof	op1 instanceof op2	If op1 is an instance of op2

Note that "==" means equal, "=" is an assignment operator.

- a = b put the value of b into memory location designated by a
- a == b test if the value of a is equal to the value of b (return true or false)

Example: relationOperator.java

```
01  public class relationOperator {
02      public static void main(String[] args) {
04          float aHeight = 166.5f;
05          float bHeight = 171.3f;
07          boolean flag;
08
09          flag = aHeight > bHeight;
10          System.out.println("Is a higher than b? : " + flag);
11
12          int aAge = 15;
13          int bAge = 15;
14
15          flag = aAge == bAge;
16          System.out.println("Are a and b same age? : " + flag);
17      }
18  }
```


Output:

```
Is a higher than b? : false
Are a and b same age? : true
```

Note that the output is only "true" or "false".

c) Logical operators

Logical operators evaluate the value and express it with true and false. There are 3 operators:

and(&&) or(||) not(!)

What does 'and' mean? And what does 'or' mean?

Let's think about a compound sentence for an example: "Apple and beef are vegetables." Does it make sense? Because 'beef' is not a vegetable, this sentence is false. Both elements have to be true to make the sentence true.

On the other hand, look at another compound sentence: "Apple or beef is vegetable." Does it make sense? Because 'or' means one of them, it make sense.

Logical operators && and ||

X	Y	X && Y	X || Y
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Logical operator !

X	!X
true	false
false	true

Note that logical operations result only in true or false.

Example: boolean flag

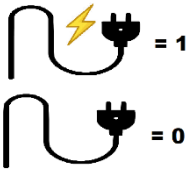
```
flag = 30 < 20;
flag = 30 && 20;
flag = 30 < 20 || 50 > 20;
flag = 30 < 20 && 80;
flag = 20 + 30 < 20 * 2
```

d) Bit operators

ACS Theory C Lesson 6 helps to understand this part.

We need to learn 'binary number system' before we start bit operators. Because we use ten numbers from 0 to 9 (0,1,2,3,4,5,6,7,8,9), it is said that we use decimal number system. But the computer doesn't understand a decimal number because it can only read 0 and 1. Why?

This is related to the internal design of computers. Everybody knows that the computers use electricity and computer is connected with cables.



- If there is no electricity, the computer recognize it as 0, and
- if there is electricity, the computer recognize it as 1.

The following table shows the bit operations on binary numbers with **and(&)**, **or(|)**, and **exclusive-or(^)** operators.

Bit operators

Operator	Way to use	Example
&	5&7	5(101) & 7(111) = 101
|	5|7	5(101) | 7(111) = 111
^	5^7	5(101) ^ 7(111) = 010

It is just same as applying logical operators onto each binary digit.

```
   11010 ==26
OR 01100 ==12
   11110 ==30
```

Calculation with bit operator (or)

e) Shift operators

Shift operators are special operators in that they move each digit of a binary number. Only integer type can be operated.

Shift operators

Operator	Way to use	Description
<<	a << n	Convert a to binary and shift n times to the left.
>>	a >> n	Convert a to binary and shift n time to the right.
>>>	a >>> n	Convert a to binary and shift n times to the right. If a is negative, convert to positive.


```
>>2  11010 == 26
     00110 == 6
```

The way shift operator works

3) Octal and hexadecimal numbers

Sometimes we need to use octal and hexadecimal numbers as well as binary numbers.

- Octal: express with 8 numbers (0, 1, 2, 3, 4, 5, 6, 7)
- Hexadecimal: express with 16 numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

How to express octal and hexadecimal numbers in Java

Number	The way to describe	Example
Octal	Put 0 before number	0324
Hexadecimal	Put 0x before number	0xACE

Of course, we can convert decimal numbers to equivalent octal, hexadecimal and binary numbers or vice versa. However Java provides a method for those conversions.

Related conversion methods use the following syntax:

```
Integer.toBinaryString(DecimalNumber);
Integer.toOctalString(DecimalNumber);
Integer.toHexString(DecimalNumber);
```

The methods are similar to functions.

Example: BinaryToHexadecimal.java

```
01  public class BinaryToHexadecimal {
02      public static void main(String[] args) {
04          int a = 100;
05          int b = 0100;
06          int c = 0x100;
07
08          System.out.println("Decimal 100 is " + a);
09          System.out.println("Decimal equivalent of octal 100 is " + b);
10          System.out.println("Decimal equivalent of Hexadecimal 100 is " + c);
11          System.out.println("Binary equivalent of decimal 100 is “
12              + Integer.toBinaryString(100));
13      }
14  }
```


Output:

```
Decimal 100 is 100
Decimal equivalent of octal 100 is 64
Decimal equivalent of hexadecimal 100 is 256
Binary equivalent of decimal 100 is 1100100
```

4) Assignment operator

This is a special operator. The expression can be simplified to be more efficient by using the assignment operators. Let's compare the ordinary equations to the equations with assignment operators.

Examples of assignment operators

Original equation	Equation with assignment operator
j = j + 5;	j += 5;
j = j − 8;	j −= 8;
j = j * 10;	j *= 10;
j = j / 12;	j /= 12;

Both equations produce the same results. When assignment operators are used, we write more efficient codes in terms of processing time.

Assignment operators

Operator	Way to use	Meaning
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

You don't have to memorize all these operators. They will become handy if you used them many times.

Example: AssignmentOperators.java

```
01      public class ShortenOperator {
02              public static void main(String[] args) {
04                      int a = 5;
05
06                      a += 5;
07                      System.out.println("a = " + a);
08
09                      a %= 4;
10                      System.out.println("a = " + a);
11
12                      a <<= 4;
13                      System.out.println("a = " + a);
14
15                      boolean b = false;
16
17                      b &= a > 2;
18                      System.out.println("b = " + b);
19
20                      b |=    a > 2;
21                      System.out.println("b = " + b);
22              }
23      }
```


Output:

```
a = 10
a = 2
a = 32
b = false
b = true
```

5) Ternary operator

Java has an operator that has 3 operands.

A ? B : C;

A and B and C are expressions where it returns B if A is true. Otherwise, it returns C.

Example: TernaryOperator.java

```
01    public class result {
02        public static void main(String[] args) {
04            int a = 10 > 5 ? 3 : 5;
05            Boolean b = 100 == 101 ? true : false;
06
07            System.out.println(a + ", " + b);
08        }
09    }
```

Output:

```
3, false
```

We have learned all operators. There is something we have to know at the end. What if we use different kinds of operators at the same time? What would be the order of calculation?

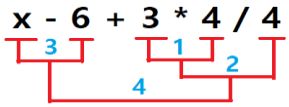
In normal math calculation of 2+3*4, we multiply first and then add. The reason is that multiplication has higher priority than addition. Similar priorities are there for Java operators.

6) Priority of Operators

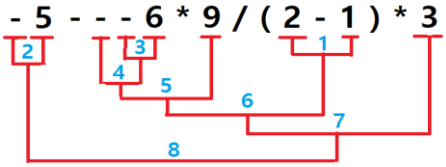
Priority (1 is the highest.)	Operator	Example	Name
1	[ ]	a[b]	Index operator
	( )	(a+b)*c	Bracket
	.	a.b	Member access operator
2	++	++a, a++	Increment operator
	--	--a, a--	Decrement operator
	+	+a	Unary + operator
	-	-a	Unary – operator
	!	!a	NOT operator
	~	~a	Bit complement operator
3	new	new className	new operator
	( )	(short)	Cast operator
4	*	a * b	Multiply operator
	/	a / b	Divide operator
	%	a % b	Modulo operator
5	+	a + b	+ operator
	-	a - b	- operator
6	<<	a << b	Shift operators
	>>	a >> b	
	>>>	a >>> b	
7	<	a < b	Relational operators
	>	a > b	
	<=	a <= b	
	>=	a >= b	
	instanceof	a instanceof b	
8	==	a == b	Equivalent operator
	!=	a != b	
9	&	a & b	Bit AND operator
10	^	a ^ b	Bit XOR operator
11	|	a | b	Bit OR operator
12	&&	a && b	Logical AND operator
13	||	a || b	Logical OR operator
14	?:	a ? b : c	Ternary operator
15	=	a = b	Assignment operator
	*=, /=, …	a *= b, a /= b, …	Shortened operator

If operators have same priority, calculate from left to right.

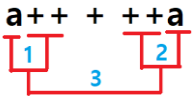
Example:



Example:



Example: Bracket go first and calculate unary operators



Let's say a = 10. What would be the answer?

7) Identifiers

Identifiers are the names we create such as the name of a variable. We can create whatever name we want but there are a few rules:

i. You can use letters, numbers, and special characters (_, $).

ii. You can start the identifier with letters and special characters but not with numbers.

iii. Reserved words are not available for an identifier.

iv. True, false, null can't be an identifier.

v. Length of an identifier does not matter.

vi. Upper and lower characters are different. (case sensitive)

"Reserved words" in rule iii. mean the key words that are already defined in Java. For example, 'int' is a reserved word meaning an integer type. So, 'int' cannot be an identifier.

Reserved words of Java

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

And there are some habits for Java programmers. The first letter of the variable name needs to be lower case; the first letter of a class needs to be upper case; the name of a method has to be lower case. If you don't follow these habits, you will not get any errors. To help other programmers understand the code easily, everybody just keep those conventions.

8) Input

All codes we have learned so far were to print something. What if we need to give some input to computers? To give some input, we need to learn two more things that we never talked about. They are "header" and "generate an object". These parts are tough to understand for now and the details will not be covered. Just memorize that two parts of code are necessary to make input.

Example: input1.java

```
01  import java.util.Scanner;
02
03  public class input1 {
04      public static void main(String[] args) {
05          int a;
06          System.out.print("Type a number: ");
07          Scanner sc = new Scanner(System.in);
08          a = sc.nextInt();
09          System.out.println(" Your number is " + a);
10      }
11  }
```

Input:

Type a number: 5

Output:

Your number is 5

To accept an input, three lines of code are essential. What are those lines? Input can be anything you type. But the computer needs to save your input in a specific type such as int, char, String, etc. You can tell computer what kind of input the user will give. There are methods to get each type of input as below.

- nextBoolean() : the type of the next input is Boolean.
- nextDouble() : the type of the next input is double.
- nextFloat() : the type of the next input is float.
- nextInt() : the type of the next input is int.
- nextLine() : the type of the next line of input is String.
- nextLong() : the type of the next input is long.
- nextShort() : the type of the next input is short.
- next().charAt(0) : the type of the next input is char.

Except nextLine(), space(" ") can separate the input.

For example,

```
int a = sc.nextInt();
int b = sc.nextInt();
```

If you type "10 11", the value of a will be 10 and the value of b will be 11.

Example: input2.java

```
01  import java.util.Scanner;
02  public class input2 {
03      public static void main(String[] args) {
04          String s;
05          int a;
06          char c;
07          double d;
08
09          Scanner sc = new Scanner(System.in);
10
11          System.out.print("write your name : ");
12          s=sc.nextLine();
13          System.out.println("your name is " + s);
14
15          System.out.print("type your age : ");
16          a=sc.nextInt();
17          System.out.println("you are " + a + " years old");
18
19          System.out.print("write a letter : ");
20          c=sc.next().charAt(0);
21          System.out.println("your letter is " + c);
22
23          System.out.print("write the number of pi as much as you know : ");
24          d=sc.nextDouble();
25          System.out.println("pi is " + d);
26      }
27  }
```


Input:

```
write your name : George Washington
type your age : 15
write a letter : A
write the number of pi as much as you know : 3.14
```

Output:

```
your name is George Washington
you are 15 years old
your letter is A
pi is 3.14
```

9) Make your program readable

We usually make programs for other people. That means your code needs to be well-organized for reading.

a) Naming an identifier

The first letter of variables and methods has to be lower case and the first letter of a class has to be upper case. If an identifier consists of more than 2 words, the first letter of words has to be upper case. A constant has to be upper case.

```
Ex1)  class Overwatch        // class name
Ex2)  int age;               // variable name
Ex3)  int ageOfSister;       // variable name with three words
Ex4)  define Pi 3.14         // constant
```

b) Spacing and changing lines

Spacing and changing lines never affect the execution of a program. They are just great ways to make the code better to read.

```
Ex)   int i=1+2;             // better to put spaces between each operator and operand
      → int i = 1 + 2;
Ex)   boolean b=(3.04*49+62)>=(23-4)/92%3*92;
      → boolean b = (3.04 * 49 + 62) >= (23 - 4) / 92 % 3 * 92;
```


In a program, there are many different types of codes such as declare, calculation, input and output etc. It is better to separate each part by changing lines.

```
Ex) // sample code to show how to separate codes
    Int         a = 10;
    Float       f = 3.22f;
    (change line)
    F += a;
    a++;
    (change line)
    System.out.println("numbers : " + a + f);
```

c) Annotation (or Comments)

If you want to take a note about the code, it is always a good idea. You may describe how the program works, the meaning of each variable etc. The annotation does not affect the execution of codes. They are considered as memos and just for understanding the code.

There are 3 ways of making annotation.

i. // annotate a line.

ii. /*……..*/ annotate from the start to an end.

iii. /**……..*/ annotate several lines and make description following Javadoc.

Example comment.java

```
01  //booking program
02  import java.util.Scanner;
03  public class comment {
04      public static void main(String[]args) {
05
06          Scanner sc = new Scanner(System.in);
07          System.out.println("If you want to book, tell me your name : ");
08          String name = sc.nextLine();   //get input in String type
09          System.out.println(name + ", you made a reservation.");
10      /*
11          System.out.println("have a nice day!");
12          System.out.println("see you next time!");
13      */
14      }
15  }
```


If there is an error in your code, you can annotate the suspicious parts of your code and try other ways until the error is fixed.

10) Homework

1. What is printed when the following code is run?

```
public class example {
        public static void main(String[] args) {
                int a = 5;
                int b = a++;
                int c = ++a;
                int d = b++;
                int e = d++ + ++a;

                System.out.println(a+", " + b + ", " + c + ", " + d + ", " + e);
        }
}
```

Output:

2. Predict the output of the following program.

```
public class example {
        public static void main(String[] args) {
                int a = 10;
                int b = 5;

                ++a;
                System.out.println(a);
                b = a++;
                System.out.println(b);
                System.out.println(a);
                a = ++b;
                System.out.println(a);
                ++b;
                a++;
                System.out.println(a + b);
        }
}
```


Output:

3. Variable a of int type is 382 * 952. Variable b of int type is 584 * 592. Use relational operator(s) to find the higher number and print it out.

Output:

363664

4. Convert decimal 492 to binary, octal, hexadecimal numbers.

Output:

492 in binary is 111101100
492 in octal is 754
492 in hexa is 1ec

5. There is an int type variable containing 200. This number will be increased by 482*92-321. Use shortened assignment operator to print the value.

Output:

44223

6. Declare two variables. The value of each variable is 30 and 29, respectively. Use ternary operator to find which variable has greater value.

Output:

30

7. The price of a ticket is five dollars. The program asks a user "How many tickets do you need?" and gets the input. Then, the program will tell the user total ticket price.

Output:

```
How many tickets do you need? 6
The total ticket price is 30 dollars.
```


8. Predict the output of the following program.

```
public class result {
	public static void main(String[] args) {
		int x = 1;
		System.out.println((true) && (3 > 4));
		System.out.println( !(x > 0) );
		System.out.println( (x != 1) || (x < 1) );
		System.out.println( (x >= 0) || !(x == 1) );
	}
}
```

Output:

9. Predict the output of the following program.

```
public class result {
        public static void main(String[] args) {
                System.out.println(4<<2);
                System.out.println(-4<<2);
                System.out.println(3&2);
                System.out.println((3^2));
                System.out.println((3|2));
        }
}
```

Output:

10. Predict the output of the following program.

```
public class result {
        public static void main(String[] args) {
                int a = 3, b = 2, c = 1, d = 4, e = 5;
                int sum1 = a * b + d / c % e;
                int sum2 = -a * (b + b) - c++ * d / (4 - 2) * e;
                int sum3 = (a - b) * 10 % 3 + --c * d - (4 - 2);

                System.out.println("sum1 = " + sum1);
                System.out.println("sum2 = " + sum2);
                System.out.println("sum3 = " + sum3);
        }
}
```

Output:

11. Find error(s) if any.

```
public class result {
        public static void main(String[] args) {
                int i = 30>25;
                boolean b1 = 42*36||24*19;
                boolean b2 = 24>2&&39%3<3;
                float f = (byte)39+20;
                String s = "1231412 + 3291321 /482332";
        }
}
```

12. I'm trying to make a registration program for a game company. Get a user's profile and print their information. The profile consists of name, age, e-mail address, and phone number.

Sample output:

```
Name : Aaron Kim
Age : 15
E-mail : Aaron@zmail.com
Phone : 4082839123
```


13. Before building a structure, a building company needs to find out the volume of that place. The building designer John who built thousands of buildings doesn't want to calculate by hand the volume of the place anymore. So, he planned to make a program that calculates the volume if a user writes width, length, height.

Sample output:

```
Width : 200
Length : 200
Height : 400
Volume : 16000000
```

14. ACE Pizza wants to provide a differentiated service to customers. They decided to make pizza in sizes that the customers want. Not like small, regular, large size. Get the radius of the pizza from customer and make that size of pizza. Write a program that gets radius from customer and tell the customer the area of the pizza and the price of the pizza (1 dollar per 1 cm radius). Assume that π is 3.14.

Sample output:

```
Welcome to ACE Pizza!
Tell me the radius of your pizza: 10
The area of your pizza is 314cm2
It is 10 dollars.
```

15. We are trying to make an ATM machine. ATM gives the money in the most efficient way, where the least number of bills should be used. Make a program that gets the amount of the money and calculates the best way to give. There are only 100, 50, 20, 5, 1 dollar billss and no cent is accepted.

Sample output:

```
how much? : 329
100 dollars : 3
50 dollars : 0
20 dollars : 1
5 dollars : 1
1 dollars : 4
```

A.C.E.

4. Conditional Statements

1) If statement

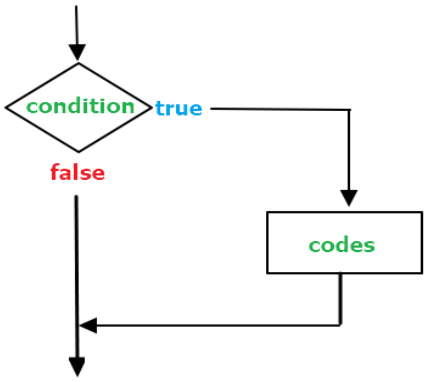
We are living in a world of choices and decisions. I don't even know how many times I make up my mind in a day. I even choose what to eat for lunch today. Eat pizza or hamburger? If you want to eat pizza, you have to go to pizza place. If you want to eat hamburger, you may go to McDonald's.

We have to choose between pizza and hamburger in order to decide the destination. This situation can be described as "If I want a pizza, then I go to pizza place." Or, "If I like hamberger better for my lunch, then I go to McDonald's."

Similarly, we have **if statement** in Java to help programmers to solve the situations of choice / decision The figure at the right shows the pictorial representation of an if statement.

The syntax of if statement is shown below:.

```
if (condition) {
        codes;
};
```



If the value of condition is true, execute the codes inside the brackets. Otherwise, skip the codes (do not execute) inside the brackets.

The condition has to reresult in true or false. Condition has no limit on it's length or complexity.

Example: ifStatement.java

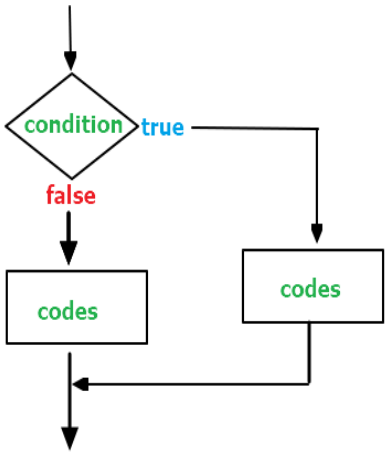
```
01  public class ifStatement {
02      public static void main(String[] args) {
04          int i = 10;
05          if( i > 5 ){
06              System.out.println("Variable i is greater than 5.");
07          }
08          if( i < 5 ){
09              System.out.println("Variable i is less than 5.");
10          }
11      }
12  }
```

Output:

```
Variable i is greater than 5.
```

This if statement only works when the condition is true. What if a programmer wants to do something when the condition is false? For those situations, if-else statement is provided in Java..

```
if (condition) {
        codes;
}
else {
        codes;
}
```



If the condition is true, the codes inside the if brackets will be executed. Otherwise, the codes inside else brackets will be executed.

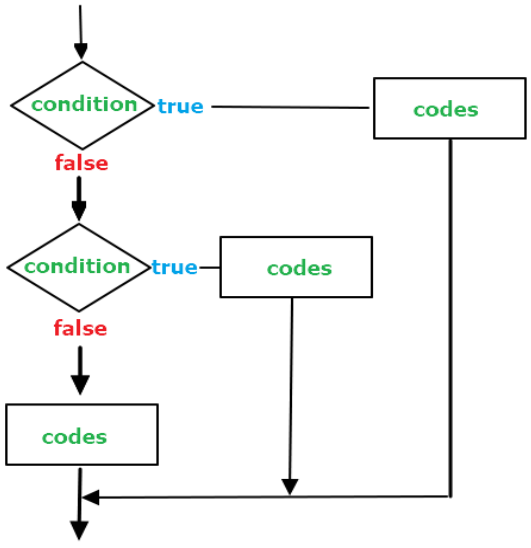
Let's construct a program which finds out if the person can drive or not depending on his/her age.

Example: IfElseStatement1.java

```
01  import java.util.Scanner;
02  public class IfElseStatement1 {
03      public static void main(String[] args) {
05          Scanner sc = new Scanner(System.in);
06          System.out.println("Tell me your age : ");
07          int age = sc.nextInt();
08          if(age < 16) {
09              System.out.println("You are not allowed to drive.");
10          }
11          else {
12              System.out.println("You are allowed to drive.");
13          }
14      }
15  }
```


We must be able to solve many different questions. The if-else statement is extended to solve multiple choice sitations as follows:

```
if (condition) {
        codes;
}
else if {
        codes;
}
else {
        codes;
}
```



You can have multiple else statements with conditions. Research an example for choosing a soda when you are in a restaurant.

Example: chooseSoda.java

```
01  import java.util.Scanner;
02  public class chooseSoda {
03      public static void main(String[] args) {
04          Scanner sc = new Scanner(System.in);
05          System.out.print("Which soda do you want to drink? (coke, sprite, Dr
pepper, fanta): ");
06          String s = sc.nextLine();
07          if (s.equals("coke")) {
08              System.out.println("Here is a coke for you.");
09          }
10          else if (s.equals("sprite")) {
11              System.out.println("Here is a sprite for you.");
12          }
13          else if (s.equals("Dr pepper")) {
14              System.out.println("Here is a Dr pepper for you.");
15          }
16          else if (s.equals("fanta")) {
17              System.out.println("Here is a fanta for you.");
18          }
19          else {
20              System.out.println("Sorry. We don't have that.");
21          }
22      }
23  }
```

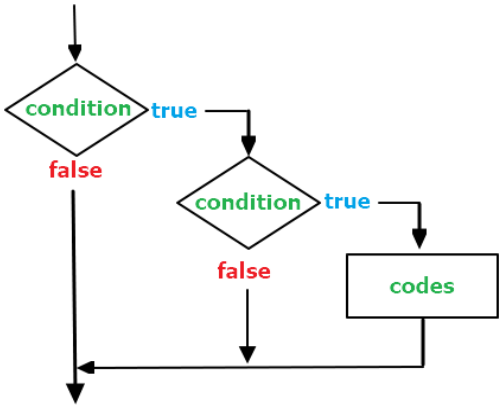

Note that we cannot use == operator to compare two different Strings. You can use "==" if the type of the variable is integer or floating point. However, **variable.equals()** is used when the type of the variable is String in lines 07, 10, 13, and 16., In other words, a String method 'equals' is invoked to compare two Strings. For now, just memorize how it was handled. We will learn it later.

Output:

```
Which soda do you want to drink?(coke, sprite, Dr pepper, fanta) : sprite
Here is a sprite for you
```

Let's think about another situation. Assume that you chose a hamburger rather than a pizza. Now you have to choose what kind of hamburger you would like. I have to choose one more time depending on the first choice. We call it **nested if statement**.

```
if (condition1) {
        if (condition2) {
                codes;
        }
}
```



Using this if statement, you can make more specific choices. Similarly, if-else statement can also be nested.

In this example, user choose fist pizza or hamburger and then choose specific kind of food.

Example: nestedIfStatement.java

```
01  import java.util.Scanner;
02  public class nestedIfStatement {
03          public static void main(String[] args) {
05                  Scanner sc = new Scanner(System.in);
06                  String food;
07
08                  System.out.print("Choose your meal (pizza or hamburger): ");
09                  food = sc.nextLine();
10
11                  if (food.equals("hamburger")) {
12                          System.out.print("What hamburger, BigMac or
    Wapper? ");
14                          food = sc.nextLine();
15                          if (food.equals("BigMac")) {
16                                  System.out.println("Let's go to Macdonald!");
17                          }
18                          else if (food.equals("Wapper")) {
19                                  System.out.println("Let's go to Burger King!");
20                          }
21                  }
22                  else if (food.equals("pizza")) {
23                          System.out.print("What pizza, Domino or Pizzahut?");
25                          food = sc.nextLine();
26                          if (food.equals("Domino")) {
27                                  System.out.println("Let's go to Domino!");
28                          }
29                          else if (food.equals("Pizzahut")) {
30                                  System.out.println("Let's go to Pizzahut!");
                            }
32                  }
34          }
35  }
```

Output:

```
Choose your meal (pizza or hamburger)? hamburger
What kind of hamburger, Bigmac or Wapper? Wapper
Let's go to Burger King!
```

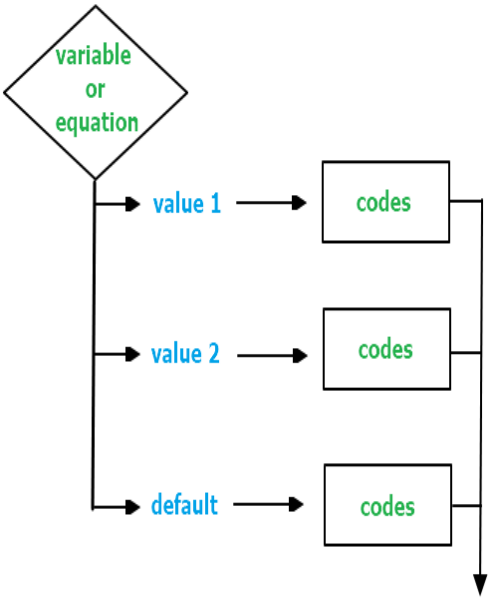
There is no specific form of nested if statement. You can use nest if in many ways to solve the problems.

2) Switch statement

If we have many choices, using if-else statement can look ambiguous and unorganized. In this case, we use switch statement rather than if else statement.

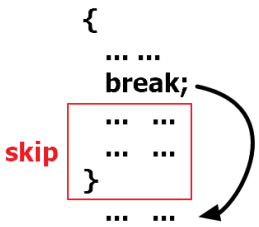
The basic form of switch statement is:

```
switch (variable) {
        case value1:
                codes;
                break;
        case value2:
                codes;
                break;
        default:
                codes;
}
```

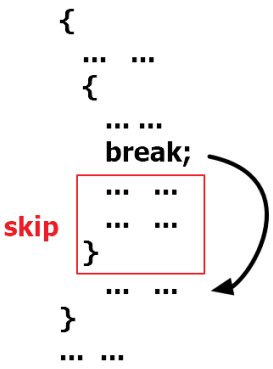


The variable part can be an expression.

The value of the variable will be the condition. The value of the variable decides which codes to execute. If the value of the variable is value1, execute the code until the 'break' statement in the following line. And get out of the switch statement. Break statement forces to get out of the brackets of switch statement, skipping all codes after 'break' until the end of the bracket. Refer to the figure at the left.



Note that the control of the program will get out of the bracket of the switch statement only. See the figure at the right.



The following example functions the same as the example we studies already except that we have more kinds of sodas.

Example: switch1.java

```
01  import java.util.Scanner;
02  public class switch1 {
03          public static void main(String[]args) {
04                  Scanner sc = new Scanner(System.in);
05                  System.out.print("Which soda do you want to drink? (coke, sprite, Dr
pepper, fanta, welchs, ginger ale) : ");
06                  String s = sc.nextLine();
07
08                  switch (s) {
09                          case "coke" :
10                                  System.out.println("Here is a coke for you");
11                                  break;
12                          case "sprite" :
13                                  System.out.println("Here is a sprite for you");
```

```
14                              break;
15                      case "Dr pepper" :
16                              System.out.println("Here is a Dr pepper for you");
17                              break;
18                      case "fanta" :
19                              System.out.println("Here is a fanta for you");
20                              break;
21                      case "welchs" :
22                              System.out.println("Here is a welchs for you");
23                              break;
24                      case "ginger ale" :
25                              System.out.println("Here is a ginger ale for you");
26                              break;
27                      default:
28                              System.out.println("Sorry, We don't have that");
29              }
30      }
31  }
```

Output:

```
Which soda do you want to drink?(coke, sprite, Dr pepper, fanta, welchs, ginger ale) : tea
Sorry, We don't have that
```


If you don't put 'break,' the computer will execute next 'case' part of the code. You may use this behavior usefully as in the following program.

Example: switch2.java

```
01  import java.util.Scanner;
02  public class example {
03          public static void main(String[]args) {
05                  Scanner sc = new Scanner(System.in);
06                  String food;
07
08                  System.out.print("Choose your favorite ingredient (apple,
mango, cucumber, carrot) : ");
10                  food = sc.nextLine();
11
12                  switch(food) {
13                          case "apple":
14                          case "mango":
15                                  System.out.println("That's a fruit!");
16                                  break;
17                          case "cucumber" :
18                          case "carrot" :
19                                  System.out.println("That's a vegetable!");
20                                  break;
```

```
21                          default :
22                                  System.out.println("error");
23                  }
26          }
27  }
```

Output:

```
Choose your favorite ingredient (apple, mango, cucumber, carrot) : mango
That's a fruit!
```

The input of this output is ‘mango’. Try other inputs too.

3) Homework

1. Predict the output.

```
public class example {
        public static void main(String[] args) {
                int x, y;
                if ( x > 10 ) {
                        if( y <= 10 ) {
                                System.out.println( x + y );
                        }
                        else {
                                System.out.println( x – y );
                        }
                }
                else {
                        System.out.println( x * y );
                }
        }
}
```

Input	output
x = 11, y = 9	
x = 9, y = 9	
x = 10, y = 10	

2. Predict the output.

```
public class example {
        public static void main(String[] args) {
                int x = 10, y = 10;
                boolean b1 = true;
                boolean b2 = false;

                if( (y == x) && !b2 ) {
                        System.out.print("A");
                }
                System.out.print("C");
                if( (b1 == false) || x < 5 ? false : true ) {
                        System.out.print("E");
                }
        }
}
```

Output:

3. Predict the output.

```
public class example {
        public static void main(String[] args) {
                int x = 4;

                switch (x) {
                        case 4:
                                System.out.println("find the trap!");
                        case 5:
                                System.out.println("there might be a trap!");
                        default :
                                System.out.println("Where is break?");
                }
        }
}
```

Output:

4. Find error(s) if any.


```
public class example {
        public static void main(String[] args) {
                int x = 10, y = 10;
                if ( x = y ) {
                        System.out.println("both of the variable have the same number");
                }
        }
}
```

5. Find error(s) if any.

```
public class example {
        public static void main(String[] args) {
                int a = 5;

                switch (a == 4) {
                        System.out.println("water");
                else {
                        System.out.println("fire");
                }
        }
}
```

6. Find error(s) if any.

```
public class example {
        public static void main(String[] args) {
                int a = 3,b = 4;

                switch( a + b ) {
                        case : 7
                                System.out.println("a+b=7");
                        case : 8
                                System.out.println("a+b=8");
                        default :
                                System.out.println("a+b="+(a+b));
                }
        }
}
```

7. Write a program.

My mom is a teacher. She never sleeps because she needs to convert score of the final test. 90 to 100 is A, 80~89 is B, 70 ~ 79 is C, 60 ~ 69 is D, rest of them are F. Let's make a program for mom to calculate in a easier way. Get a score as an input and convert to A,B,C,D,F.

Input:

```
85
```

Output:

```
B
```

8. Write a program.

A, B, C are best friends. One day, they went out to have spicy food. The spicy level of the food is from 1 to 100. The restaurant wanted each person to pick three numbers between 1 and 100. Then try the spicy level of the middle number. Build a program to find the second spicy level.

Input:

30 40 15

Output:

30

9. Write a program.

There are five students taking vocab quiz. If they get less than 16 out of 20, they have to retake the quiz. Write a program that prints out the score of the student who need to take the quiz again.

Input: five integer numbers of the vocab quiz score (0 to 20)

Output: all the score of students who didn't pass the quiz

5. Loop statements

Let's go back to the first program. Our first program was printing "Hello world!". Now, let's make a program that prints "Hello world!" 100 times.

```
public class example {
        public static void main(String[] args) {
            System.out.println("Hello world!");
            System.out.println("Hello world!");
            System.out.println("Hello world!");
            ……
}
```

There needs to be 100 lines of System.*out*.println("Hello world!") in this program. What if we have to print that 1000 times?

In this chapter, we will learn **loop** that makes this code in less than 10 lines. A loop is to repeat a particular code again. The mechanism of a loop is like this

The loop will repeat a group of codes. The programmer have to decide how many time you are going to repeat the group of codes. We have three different loop statements. Before loop statement is used, we have to find out the **pattern** of the program. Then loop statement will help you make your code more logical and concise.

1) While loop.

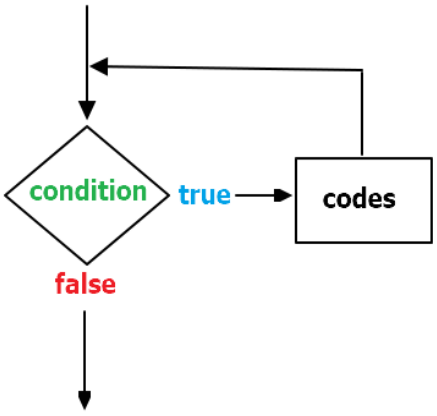
A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition (Boolean expression) is true. Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.

When executing, if the condition (Boolean expression) result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

a) Regular while loop

The syntax of while statement is:

```
while (condition) {
        statements (codes);
}
```



Again, the followings are the steps of the while loop.

- Step1. Check if the value of the condition is true.
- Step2. If the condition is true, execute the code in while loop, if the condition is false, escape from the loop.
- Step3. Do step1, step2 until escape from the loop.

Example: whileLoop1.java

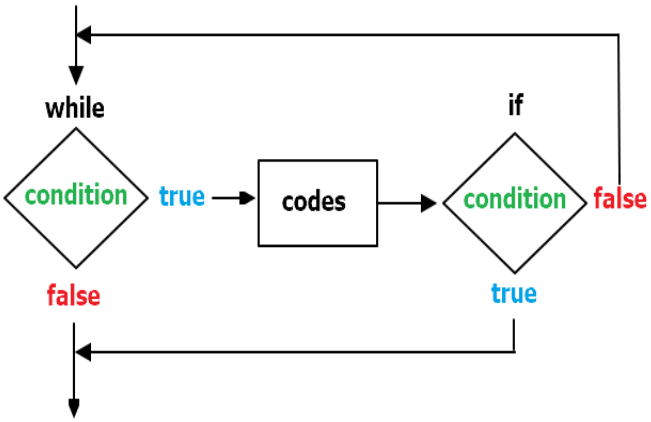
```
01  public class whileLoop1 {
02      public static void main(String[] args) {
03          int times = 0;
04
05          while(times < 100) {
06              System.out.println("Hello world!");
07              times++;
08          }
09      }
10  }
```

Output:

```
Hello world!
Hello world!
Hello world!
...
Hello world!
```

b) A while loop with true condition

If the condition is always true, it will be repeated forever. To escape from the while loop when the condition is true, we use a 'break' statement. The break statement is the same as in switch statement. The following codes show the while statement with 'true' keyword.




```
while (true) {
    codes
    if (condition) {
        breal;
    }
}
```

In the if statement, if the value of the condition is true, then break code will be executed. Which means, the program will escape from the 'while loop'.

The example program below is a donation program. Input of the program is the amount of money for donation. Until they get more donation than their target amount, this program will keep getting donation.

Example: whileLoop2.java

```
01  import java.util.Scanner;
02  public class whileLoop2 {
03      public static void main(String[] args) {
04          int maxMoney = 1000;
05          int donate;
06          int donated = 0;
07          Scanner sc = new Scanner(System.in);
08
09          while(true) {
10              if (donated >= maxMoney) {
11                  break;
12              }
13              System.out.println("Please donate to save children! (" +
    donated + "/" + maxMoney + ")");
14              System.out.print("How much are you going to donate? : ");
15              donate = sc.nextInt();
16              donated += donate;
17              System.out.println("Thank you for your donation!\n");
18          }
19          System.out.println("Donation has successfully completed!");
20      }
21  }
```

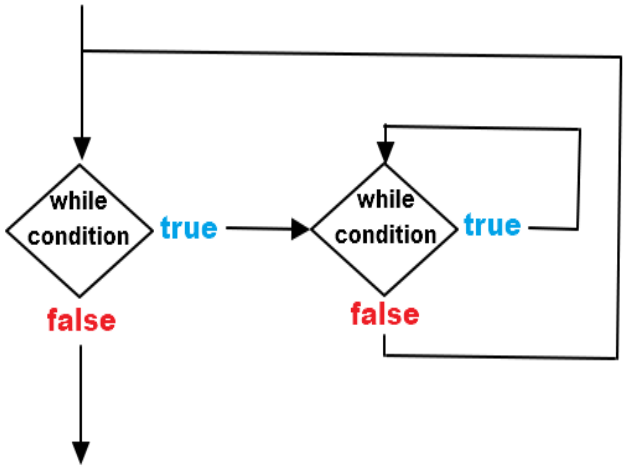

Output:

```
Please donate to save children! (0/1000)
How much are you going to donate? : 50
Thank you for your donation!

Please donate to save children! (50/1000)
How much are you going to donate? : 500
Thank you for your donation!

Please donate to save children! (550/1000)
how much are you going to donate? : 600
Thank you for your donation!

Donation has successfully completed!
```

Making appropriate condition in while loop is very important. Sometimes using infinite loop is more efficient than a particular condition. You can make condition with multiple variables by using &&, || operators.

c) Nested while loop

```
while (condition) {
        while (condition) {
                codes;
        }
        Codes;
}
```



There is another while loop in a while loop. When you become familiar with loops, you have to find out the differences between nested loop and multiple loops.

This is an example of nested while loop. If the user put a number 'N', the program will calculate the sum until N(1+2+…N). The program will repeat this until the user put 0 as an input.

Example: nestedWhileLoop.java

```
01  import java.util.Scanner;
02  public class nestedWhileLoop {
03          public static void main(String[] args) {
04                  int number = 1;
05                  int sum = 0;
06                  Scanner sc = new Scanner(System.in);
07
08                  while (true) {
09                          sum=0;
10                          System.out.print("Tell me the number (to exit write 0) : ");
11                          number = sc.nextInt();
12                          if (number == 0) {
13                                  break;
14                          }
15                          while(number != 0) {
16                                  sum += number;
17                                  number--;
18                          }
19                          System.out.println("The sum of the numbers = " + sum);
20
21                  }
22                  System.out.println("The program is over.");
23          }
24  }
```


Output:

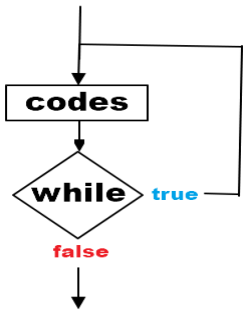
```
Tell me the number (to exit write 0) : 9
The sum of the numbers = 45
```

```
Tell me the number(to exit write 0) : 3
The sum of the numbers = 6
Tell me the number(to exit write 0) : 1
The sum of the numbers = 1
Tell me the number(to exit write 0) : 0
The program is over.
```

The first “while loop” makes this program to continue until it gets 0. The second “while loop” solves the problem. Try different numbers to understand this program.

2) do-while loop

If the condition of the value of a "while loop" is false, then the loop won't be executed. But what if you want to **execute the loop at least once**? In this case, we use "do-while" loop.




```
do {
        codes;
} while (condition)
```

This is the sequence of the do-while loop.

- Step 1. Execute a code in "do" bracket.
- Step 2. Check the condition of while loop. If it's true, execute the code inside do bracket.
- Step 3 The "do-while loop" executes the code at least once. Then, check the condition later.

Let's see the difference between while loop and do-while loop with this example.

Example: doWhileLoop.java

```
01  import java.util.Scanner;
02  public class doWhileLoop {
03          public static void main(String[] args) {
04
05                  int whileInt = 0;
06                  int doWhileInt = 0;
07
08                  while( whileInt > 0 ) {
09                          System.out.println("Execute while loop");
10                  }
11
12                  do {
13                          System.out.println("Execute do-while loop");
14                  } while (doWhileInt > 0);
15          }
16  }
```

3) "for" loop

a) Simple "for" loop

If you want to decide how many times you want to repeat the loop, "for loop" would be a good choice. Condition of "for loop" is in more detail than while loop.

```
for (initialization; condition; afterthought) {
        codes;
}
```

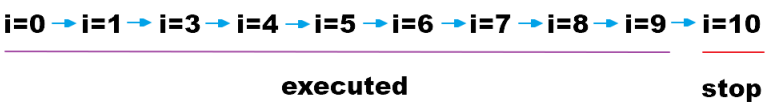
- Initialization : initialize a variable
- Condition : when the condition is true, repeat the loop
- Afterthought : in every cycle, execute this equation

These three elements tells how many time this loop will work. Here is an example of a for loop

```
for (int i = 0; i < 10; i++) {
        system.out.println("Hello world!");
}
```

How many times do you think this loop will repeat?

- Initialization : int i = 0 when the loop starts, variable i is 0
- Condition : i < 10 repeat this loop when i is less than 10
- Afterthought : every cycle of loop, increase i by 1



So, this for loop will repeat the code 10 times.

For loop will make you easy to understand how many times this loop will run. There are many ways to apply 'for loop.'

Example: Describe how many times each for loop will run.

i. for (int i = 5; i > 0; i--)
ii. for (int i = 0; i < 10; i += 2)
iii. for (int i = 0; i == 10; i += 3)

The variable "i" will disappear after escaping from the for loop. So you can declare and initialize "i" again after the for loop.

Example: forLoop.java

```
01  import java.util.Scanner;
02  public class forLoop {
03      public static void main(String[]args) {
04
05          for(int i = 0;i < 5; i++) {
06              System.out.println("hello");
07          }
08      }
09  }
```

Output:

```
Hello
hello
hello
hello
hello
```

b) Nested "for" loop

The for loop can be nested as well. But be careful. You usually need another variable for the other for loop. One example of nested loop is shown:

```
for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        system.out.println("Hi ACE");
```

How many times do you think this "for loop" will print "hello"? The table below shows all the steps of this for loop. It will print "hello" 100 times.

i = 0, j = 0, printed 1 times	i = 5, j = 0, printed 51 times	i = 2, j = 5, printed 26 times	i = 7, j = 5, printed 76 times
i = 0, j = 1, printed 2 times	i = 5, j = 1, printed 52 times	i = 2, j = 6, printed 27 times	i = 7, j = 6, printed 77 times
i = 0, j = 2, printed 3 times	i = 5, j = 2, printed 53 times	i = 2, j = 7, printed 28 times	i = 7, j = 7, printed 78 times
i = 0, j = 3, printed 4 times	i = 5, j = 3, printed 54 times	i = 2, j = 8, printed 29 times	i = 7, j = 8, printed 79 times
i = 0, j = 4, printed 5 times	i = 5, j = 4, printed 55 times	i = 2, j = 9, printed 30 times	i = 7, j = 9, printed 80 times
i = 0, j = 5, printed 6 times	i = 5, j = 5, printed 56 times	i = 3, j = 0, printed 31 times	i = 8, j = 0, printed 81 times
i = 0, j = 6, printed 7 times	i = 5, j = 6, printed 57 times	i = 3, j = 1, printed 32 times	i = 8, j = 1, printed 82 times
i = 0, j = 7, printed 8 times	i = 5, j = 7, printed 58 times	i = 3, j = 2, printed 33 times	i = 8, j = 2, printed 83 times
i = 0, j = 8, printed 9 times	i = 5, j = 8, printed 59 times	i = 3, j = 3, printed 34 times	i = 8, j = 3, printed 84 times
i = 0, j = 9, printed 10 times	i = 5, j = 9, printed 60 times	i = 3, j = 4, printed 35 times	i = 8, j = 4, printed 85 times
i = 1, j = 0, printed 11 times	i = 6, j = 0, printed 61 times	i = 3, j = 5, printed 36 times	i = 8, j = 5, printed 86 times
i = 1, j = 1, printed 12 times	i = 6, j = 1, printed 62 times	i = 3, j = 6, printed 37 times	i = 8, j = 6, printed 87 times
i = 1, j = 2, printed 13 times	i = 6, j = 2, printed 63 times	i = 3, j = 7, printed 38 times	i = 8, j = 7, printed 88 times
i = 1, j = 3, printed 14 times	i = 6, j = 3, printed 64 times	i = 3, j = 8, printed 39 times	i = 8, j = 8, printed 89 times
i = 1, j = 4, printed 15 times	i = 6, j = 4, printed 65 times	i = 3, j = 9, printed 40 times	i = 8, j = 9, printed 90 times
i = 1, j = 5, printed 16 times	i = 6, j = 5, printed 66 times	i = 4, j = 0, printed 41 times	i = 9, j = 0, printed 91 times
i = 1, j = 6, printed 17 times	i = 6, j = 6, printed 67 times	i = 4, j = 1, printed 42 times	i = 9, j = 1, printed 92 times
i = 1, j = 7, printed 18 times	i = 6, j = 7, printed 68 times	i = 4, j = 2, printed 43 times	i = 9, j = 2, printed 93 times
i = 1, j = 8, printed 19 times	i = 6, j = 8, printed 69 times	i = 4, j = 3, printed 44 times	i = 9, j = 3, printed 94 times
i = 1, j = 9, printed 20 times	i = 6, j = 9, printed 70 times	i = 4, j = 4, printed 45 times	i = 9, j = 4, printed 95 times
i = 2, j = 0, printed 21 times	i = 7, j = 0, printed 71 times	i = 4, j = 5, printed 46 times	i = 9, j = 5, printed 96 times
i = 2, j = 1, printed 22 times	i = 7, j = 1, printed 72 times	i = 4, j = 6, printed 47 times	i = 9, j = 6, printed 97 times
i = 2, j = 2, printed 23 times	i = 7, j = 2, printed 73 times	i = 4, j = 7, printed 48 times	i = 9, j = 7, printed 98 times
i = 2, j = 3, printed 24 times	i = 7, j = 3, printed 74 times	i = 4, j = 8, printed 49 times	i = 9, j = 8, printed 99 times
i = 2, j = 4, printed 25 times	i = 7, j = 4, printed 75 times	i = 4, j = 9, printed 50 times	i = 9, j = 9, printed 100 times

4) Continue statement

The continue statement is opposite to break statement. The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes control to immediately jump to the update statement.
- In a while loop or do/while loop, control immediately jumps to the Boolean expression.

This example is a program that adds all the odd numbers in the range of 1 to 10.

Example: continueStatement.java

```
01  public class continueStatement {
02      public static void main(String[] args) {
03          int a=0;
04          int sum = 0;
05
06          while(a < 10) {
07              a++;
08              if( a % 2 == 0 ) {
09                  continue;
10              sum += a;
11          }
12          System.out.println(sum);
13      }
14  }
```


Output:

```
25
```

Now we have learned three different types of loop. When you are solving a question, you follow these steps:

- Step 1. Find the patterns of the problem.
- Step 2. Choose which type of loop you will use.
- Step 3. Consider what kind of condition you will use to end the loop.
- Step 4. Consider what codes you will repeat to solve the problem.

5) Homework

1. What is the output of this program?

```
public class example {
        public static void main(String[] args) {
                int times = 0;
                int sum = 0;
                while(times<10) {
                        sum+=times*2;
                        times++;
                }
                System.out.println(sum);
        }
}
```

Answer:

2. What is the output of this program?

```
public class example {
        public static void main(String[] args) {
                int i = -1;
                int sum=0;

                do {
                        sum += 1;
                        i++;
                } while (i < 5);
                System.out.println(sum);
        }
}
```


Answer:

3. What is the output of this program?

```
public class example {
        public static void main(String[] args) {
                int i = 10;
                int sum=0;

                for(int k = 0; k < 10; k++) {
                        sum+= k + i;
                }
                System.out.println(sum);
        }
}
```

Answer

4. What is the output of this program?

```
public class example {
        public static void main(String[] args) {
                for(int i=0; i< 5; i++) {
                        for(int j=5; j>0; j--) {
                                System.out.print("*");
                        }
                        System.out.println("");
                }
        }
}
```


Answer:

5. What is the output of this program?

```
public class example {
        public static void main(String[]args) {

                int a = 0;
                int b = 0;
                int c = 0;
                int d = 0;

                for(int i=0; i< 10; i++) {
                        a+=i;
                }
                System.out.println(a);

                for(int i=0; i<10; i++) {
                        b--;
                        for(int j=0; j<10; j++) {
                                b++;
                        }
                }
                System.out.println(b);

                for(int i=0;i<5; i++) {
                        d=0;
                        c++;
                        while(d<10) {
                                c++;
                                d++;
                        }
                }
                System.out.println(c);
        }
}
```


Answer:

6. Find programming error(s) if any.

```
public class example {
        public static void main(String[]args) {

                int a=0;
                int sum = 0;

                while(a<10) {
                        if(a%2==0) {
                                continue;
                        }
                        a++;
                        sum+=a;
                }
                System.out.println(sum);
        }
}
```

Answer:

7. Find programming error(s) if any.

```
public class example {
        public static void main(String[]args) {

                int a=0;
                int sum = 0;

                for(int i=0; i< 5; i++) {
                        for(int j=0; j<5; i++) {
                                System.out.println("Hello");
                        }
                }
        }
}
```

Answer:

8. Write a program.

Input is an positive integer number ‘N’. Print all numbers from 1 to N, one number per line.

9. Write a program.

Input is an positive integer number ‘N’. Print all numbers from N to 1, one number per line.

10. Write a program.

My sister is learning the rule of multiplication. To help her, I'm planning to make a multiplication table.

Output:

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
......
9 X 9 = 81
```

11. Write a program.

Print the asterisks and generate the output as below by using loops.

Output:

```
    *
   **
  ***
 ****
*****
```

12. Write a program.

Input of this program is a positive integer ‘N’. From 1 to N, find the sum of all odd numbers and doubles of all even numbers.

Hint) For N = 5, Sum = 1 + 4 + 3 + 8 + 5 = 21.

13. Write a program.

Andre, graduated from UC Berkley, told the students how to study. He always studied from the first page. Let's say he studied 3 pages a day. On the first day, he studied pages 1 to 3; on the second day, pages 1 to 6; on the third day, pages 1 to 9; and so on. Write a program that calculates how many pages a student will study in a specific period of days. The first input is the number of pages that a student will study per day. The second input is how many days a student will study.

14. Write a program.

John is a game developer, who is developing a game with Egyptian background. He wants to describe pyramids of different height using asterisks. The input is a positive integer for the height of the pyramid.

Sample input:

```
4
```

Sample output:

```
   *
  ***
 *****
*******
```

15. Write a program.

ACE Pizza has difficulty in reserving seats for customer calls. When a customer gives a call for reservation, sometimes he/she tells the date, but not the day. The restaurant only opens during the week. So George always has to look for a calendar if the day is available or not. He decides to make a program, which tells the day when the call receiver puts a date of year 2018 as input.

January first of year 2018 is Monday. January, March, May, July, August, October, December has 31 days. April, June, September, November has 30 days. And February has 28 days.

Use switch statement.

Input:

Inputs are month and day of year 2018 for reservation. There is a space between them.

Output

Output is the day of booking.date. It is one of SUN, MON, TUE, WED, THU, FRI, and SAT.

Sample input:

1 3

Output:

WED

Nine pages from here including this page are left intentionally blank to provide enough space for sketching the program blueprint.

A.C.E.

A.C.E.

A.C.E.

A.C.E.

A.C.E.

A.C.E.

A.C.E.

A.C.E.

6. Arrays

1) What is an array?

We have written a lot of programs. In most cases, less than ten variables were declared. What if we need more than that?

Imagine that we need to make variables of colors.

```
String color1 = "red";
String color2 = "yellow";
String color3 = "green";          ← Variables for 139 colors
...
String color139 = "purple";
```

Is there a way to group variables that are in a same category? Yes, an array can help to organize as a group. An array has three parts: data, index, and type. A String type array "color" can be formulated. It contains all colors above.

Index always starts from 0. If you access index 0 of color array, you will get "red". For 139 variables of colors, the last index of color array is 138.

To use an array, we should declare the array just like other variables. An array can be declared in two ways.

```
type ArrayName[];
type[] ArrayName;
```

After declaring an array, we should do two more steps. One is initializing the array and the other is creating an object. We will not learn about creating objects for now. So just think 'create an array' defines the size and the type of the array and make the array ready to use.

To declare an array, the syntax below is used:

```
name = new type[size];
```

We can declare and create an array at the same time as below:

```
type name[] = new type[size];
type[] name = new type[size];
```

So if you want to declare and create the array "color," it will look like this:

```
String color[] = new String[139];
```

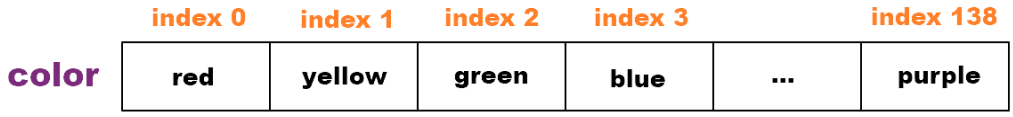
It is similar to Scanner sc = new Scanner(System.in)? Both of them created with the same concept. But let's talk about it in the next level. We will just use the syntax as it is.

There are two different ways to initialize an array.

i. Initialize after declaration and creation

```
String color[] = new String[139];
color[0] = “red”;
color[1] = “yellow”;
color[2] = “green”;
…
color[138] = “purple”;
```

In the first line, declared a String type array “color” and created it with 139 size. Second line means the data of index 0 of the array “color” is “red”. Just like this, index 1 of the array “color” is yellow. After initializing, the array will look like this.



ii. Declare and initialize at the same time

We can declare and initialize at the same time just like general variables.

```
type name[] = {data1, data2, …, dataN};
```

This time, you don’t have to decide the size because the number of data will tell the size. With color example, this code will be like this.

```
String color[] = {“red”, “yellow”, “green”, …, “purple”};
```


If you don’t initialize a variable, you can’t use that variable. But even if we didn’t initialize an array, the compiler puts default value in each index.

Default value depending on the types of an array

Type	Default value
byte, short, int, long	0
float, double	0.0
char	‘’
boolean	false
Reference data type	null

a) The way of accessing an array.

You can access an array by using an index of the array. An array has at least one index. And the index always starts from 0.

Let's study the array index by an example program. This program has an array containing 5 colors. We want to access each index and print the color.

Example: PrintColor.java

```
01  public class PrintColor {
02      public static void main(String[] args) {
04          String color[] = {"red", "yellow", "green", "blue", "white"};
05          System.out.println(color[0]);
06          System.out.println(color[1]);
07          System.out.println(color[2]);
08          System.out.println(color[3]);
09          System.out.println(color[4]);
10      }
11  }
```

Output:

```
red
yellow
green
blue
white
```


In this program, we accessed each index of the array. To take advantage of an array, we can use for loop to make it easier. The program below works exactly the same as above.

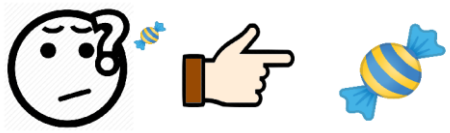
Example: printColor2.java

```
public class printColor2 {
    public static void main(String[]args) {
        String color[] = {"red", "yellow", "green", "blue", "white"};

        for(int i=0; i<5; i++) {
            System.out.println(color[i]);
        }
    }
}
```

Now we can access each color with index. Using for loop, we can access all indexes easily. It helps to control complicated problems with patterns.

What if we want access to an array, not to an index of an array? To understand this, we have to understand the difference between ways of storing a variable and an array into the memory. Let's compare them.

b) Variable and array



If someone asks where to get a candy, you may just give a candy. (variable)

Array mechanism:

If someone asks where to get a candy, you may tell the address of a candy shop. (array)

The point here is that variable has the data, whereas array has the address of the data.

For each memory location, we associate two concepts: address and data. The computer has huge memory, so an address is necessary to find the exact location and get the data.

For the variables in memory, memory address is not important. It will directly save the **data** in the name of a variable.

```
String color1 = “red”;
String color2 = “yellow”;
String color3 = “green”;
String color4 = “blue”;
String color5 = “white”;
String color6 = “black”;
String color7 = “pink”;
…
String color139 = “purple”;
```


address	data	
13241	red	color1
13242	In use	
13243	yellow	color2
13244	In use	
13245	In use	
13246	green	color3
13247	blue	color4
13248	In use	
13249	white	color5

On the other hand, for the array in memory, the data of an array is the **address** of a memory location which is the start of the first index of the array.

```
String color[] = {
        “red”,
        “yellow”,
        “green”,
        “blue”,
        “white”,
        “black”,
}
```

address	data	
11032	48320	color
…		
48320	red	color[0]
48321	yellow	color[1]
48322	green	color[2]
48323	blue	color[3]
48324	white	color[4]
48325	black	color[5]

c) Enhanced "for" loop

Array is a great tool to organize data (search or sort data in other words). Loops always follow arrays. To make it more efficient, we have another for loop to access an array. We call it "enhanced for loop."

```
for (type variableName : arrayName) {
    codes;
}
```

The enhanced for loop enables access an array from first index to last index. Below is an example of using the enhanced for loop:

```
int sum = 0;
int numbers[] = {1, 2, 3, 4, 5};

for (int x : numbers) {
        sum += x;
}
```

This code is for adding all number in the array "numbers". The integer variable x is going to be the value of each index of "numbers". We can easily figure out that x automatically changes from 0 to 4.

After this loop, the value in "sum" will be 1+2+3+4+5 = 15.

Let's practice with another example of using enhanced for loop. In this example, the program find the biggest number in an array.

Example: enhancedForLoop.java

```
01  public class enhancedForLoop {
02      public static void main(String[] args) {
03
04          int numbers[] = {93,132,491,512,123,592,131,392,11};
05          int max = 0;
06
07          for(int x : numbers) {
08              if (x > max) {
09                  max=x;
10              }
11          }
12          System.out.println("The biggest number in this array is " + max);
13      }
14  }
```

Output:

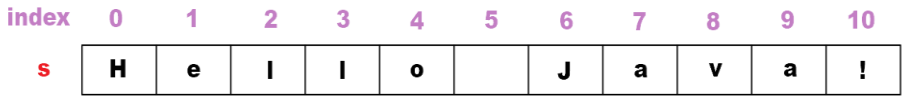
```
The biggest number in this array is 592
```

2) String

String type is just exactly the same as array. Let's see this String variable.

String s = "Hello Java!";

The structure of String s will be like this



Isn't it just look the same as an array? At each index of a String type, there exists only one letter. We can use String type just like an array. In this section, you will learn useful method of String type.

String methods

Method	Definition	Example
startWith	If a String type variable starts with selected letter, return true. Else return false.	String s = "ACE"; Boolean check=s.startWith("A"); //check = true
endWith	If a String type variable ends with selected letter, return true. Else return false.	String s = "ACE"; Boolean check = s.endWith("E"); //check = true
equals	Compare two String type variables, if they are same, return true, if not, return false.	String s1= "Aaron",s2 = "Sam"; Boolean check = s1.equals(s2); //check = false

indexOf	Return the index of a String variable which has the selected letter.	String s = "ACE";
		int check = s.indexOf("C");
		//check = 1;

length	Return the length of a String.	String s = "Hello World";
		int check = s.length();
		//check = 11

lastIndexOf	Return the index of a String variable which has selected letter for the last.	String s = "afreeca"; Int check = s.lastIndexOf("a"); //check = 6
substring	Select the range of a string and return it. (range : Start to end+1)	String s = "hello World"; String check = s.substring(0,5); //check = "hello"
toLowerCase	Change the String to lower case	String s1 = "ACE"; String check = s1.toLowerCase(); //check = "ace"
toUpperCase	Change the String to Upper case	String s = "ace"; String check = s.toUpperCase(); //check = "ACE"
trim	Eliminate all whitespace in the string.	String s = " a c e "; String check = s.trim(); //check = "ace"
contains	If a string contains another string, return true, if not, return false.	String s1 = "applepie"; Boolean check = s1.contains("pie"); //check = true
charAt	Find the letter of an index and return it as char type.	String s = "ACE"; Char check = s.charAt(1); //check = 'C'
split	Split a string by a letter and put each part of string into an array	String s = "Now I see the light"; String check[] = s.split(" "); //check[0] = "Now"

```
//check[1] = “I”
//check[2] = “see”
//check[3] = “the”
//check[4] = “light”
```

You can use these method to solve problems. This is an example of finding the longest word in a sentence.

Example: longestWord.java

```
01  import java.util.Scanner;
02  public class longestWord {
03      public static void main(String[] args) {
04
05          Scanner sc = new Scanner(System.in);
06          String s = sc.nextLine();
07
08          String words[] = s.split(" ");
09          String longest_word="";
10          int max=0;
11
12          for(String x : words) {
13              if( x.length() > max ) {
14                  max = x.length();
15                  longest_word = x;
16              }
17          }
18
19          System.out.println("the longest word in the sentence is : " +
longest_word);
20      }
21  }
```


Input:

All my life has been a series of doors in my face and then suddenly I bump in to you

Output:

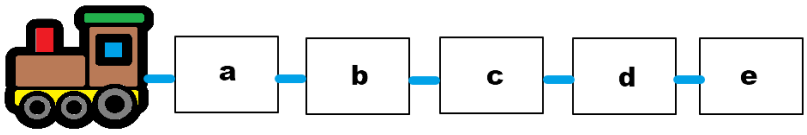
the longest word in the sentence is : suddenly

As you see in the example above, if you use the methods accordingly, you can save many lines. Before you write codes to solve a problem, research what methods can help your logic.

3) 2D array

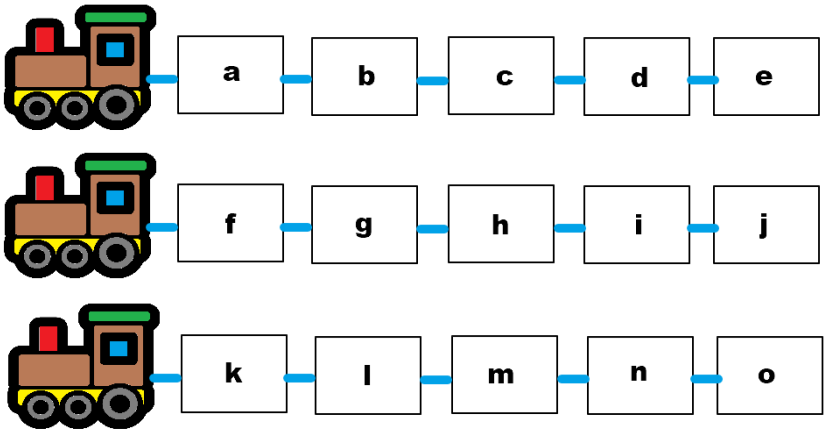
Array is like a train. Each box of train contains data. Take below sentence as an example:

```
String alphabet[] = {“a”, “b”, “c”, “d”, “e”};
```



Alphabet train

What if we have multiple trains?

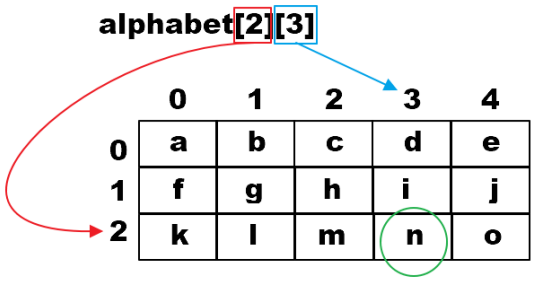


These three trains implies the concept of two dimensional array. The code below declares and initializes three trains of alphabets:

```
String alphabet[][] = {
        {“a”, “b”, “c”, “d”, “e”},
        {“f”, “g”, “h”, “i”, “j”},
        {“k”, “l”, “m”, “n”, “o”}
};
```

Now, there are two brackets next to the name of the variable. First bracket tells how many trains we have. (mathematically y-coordinate, vertical) Second bracket tells now many boxes each train has. (mathematically x-coordinate, horizontal)

Now we have two coordinates (two brackets). What does alphabet[2][3] represent?



Index of train[y] starts from the top. First bracket ‘[2]’ means index 2 of the vertical. Second bracket ‘[3]’ means index 3 of horizontal. Now, alphabet[2][3] represents “n”.

Below shows a program using 2D array. There are 4 students each of whom took 3 different exams. The program will find who has the best score out of the sum of 4 exams.

Example: highestScore.java

```
01  public class highestScore {
02      public static void main(String[] args) {
03          int scores[][] = {
04              {94,78,82,93},          //student 1
05              {93,79,88,90},          //student 2
06              {89,85,95,90},          //student 3
07              {99,90,95,100}          //student 4
08          };
09          int max=0, index=0, sum=0;
10
11          for(int i = 0; i < 4; i++) {
12              sum=0;
13              for(int j = 0; j < 3; j++) {
14                  sum += scores[i][j];
15              }
16              if(sum > max) {
17                  max=sum;
18                  index=i;
19              }
20          }
21          System.out.println("student number "+ (index+1) + " got the
highest score!");
22      }
23  }
```


Output:

```
student number 4 got the highest score
```

Just like the one dimensional arrays, we have to declare and create/initialize two dimensional arrays to use.

i. Declare and create a 2D array

type name[][] = new type[][];

ii. Declare and initialize a 2D array

type name[][] = { {v1, v2, …}, {…}, {…}, …};

Following the concept above, we can make n^{th} dimensional arrays, too.

You can find how many data are in an array by using arrayName.length method. If you want to use this method in 2D array, there are two different lengths. The length of a row and the number of rows.

2DarrayName[].length	**the length of a row**
2DarrayName.length	**the number of rows**

4) Homework

1. What is the output of this program?

```
public class example {
        public static void main(String[] args) {

                int array1[] = new int [5];
                array1[0] = 1;
                array1[1] = 2;
                array1[2] = 3;
                array1[3] = 4;
                array1[4] = 5;

                for(int i=0;i<array1.length; i++) {
                        System.out.println(array1[4-i]);
                }
        }
}
```

2. What is the output of this program?

```
public class example {
        public static void main(String[] args) {

                String a="hello I want to be your friend";
                String b = a.substring(0, 5);
                b=b.toUpperCase();
                String c[] = a.split(" ");

                System.out.println(b);

                for(int i = 0; i < c.length ; i++) {
                        System.out.println(c[i]);
                }

                System.out.println(c[0].equals(b));
        }
}
```

3. What is the output of this program?

```
public class example {
        public static void main(String[] args) {
                int array2[][] = new int [3][5];

                for(int i=0; i<array2[1].length; i++) {
                        array2[1][i] = i*3;
                }

                for(int i = 0; i<array2[2].length; i++) {
                        array2[2][i] = 5%(i+1);
                }

                for(int i=0;i<array2.length;i++) {
                        for(int j = 0; j<array2[0].length; j++) {
                                System.out.print(array2[i][j]);
                        }
                        System.out.println("");
                }
        }
}
```

4. Find error(s) if any.

```
public class example {
        public static void main(String[] args) {
                int a[][] = {{0,1,2,3,4},{"zero","one","two","three","four"."five"}};

                for(int i=0; i<2; i++) {
                        for(int j=0; j<5; j++) {
                                System.out.println(a[i][j]);
                        }
                }
        }
}
```

5. Find error(s) if any.

```
public class example {
        public static void main(String[] args) {
                int tmp;
                int numbers[]= {4213, 62, 262, 453, 1534, 451};

                for(int i=0; i< 6; i++) {
                        tmp = numbers[i];
                        numbers[i] = numbers[i+1];
                        numbers[i+1] = tmp;
                }
                for(int i= 0; i< 6; i++) {
                        System.out.println(numbers[i]);
                }
        }
}
```

6. Write a program.

Brian is an English test scorer. He scored very fast in listening, reading part. But it took so long time to find out how many words are in the essay writing part. He decided to make a program that counts the number of words. Input is a sentence (String) and output is the number of words. Use String.split method.

Sample input:

```
Snow glows white on the mountain tonight. Not a footprint to be seen
```

Sample output:

```
13
```

7. Write a program.

Sam works in a bakery. The senior patisserie always tell him what to bake for today in a weird way. Today he needs to bake 0 to 9 numbered cookies. The senior patisserie told him how many cookies he should bake. He gives three numbers with three digits, and Sam needs to find its product. Sam needs to count each digit number so that he will be able to know how many each numbered cookies must be baked. Make a program that tells how many numbered cookies he should bake for each number. Inputs are three numbers with three digits and outputs:are the numbers of numbered cookies he has to bake for each number (0 to 9)

Sample input:

123 456 789

Sample output:

0
0
2
2
3
1
0
0
0
0

8. Write a program.

Aaron is a producer of a TV quiz show. This OX quiz show is a little bit different from other quiz show. There are 15 questions, one point per question. The more the participant matches the question, the more extra point the participant gets. If it is incorrect, the additional points will be reset.

For example:

O: correct, X: incorrect

OOOXOXXOXOOOOOO	1+2+3+1+1+1+2+3+4+5+6= 29 (points)
OXOXOXOXOXOXOXO	1+1+1+1+1+1+1+1+1= 8 (points)

Write a program that scores the participant. Inputs are the score of 15 questions. Marked "O" if it's correct and marked "X" if it's incorrect. Separated by "," and output is total points.

Sample input:

O,X,O,O,X,O,O,O,O,X,O,O,O,O,O

Sample output:

29

9. Write a program.

John is an intern of my company. Even though I told him 10 times how to organize files in ascending order, he still makes mistakes! I decided to make a program that checks if the files are organized or not. File numbers should be in ascending order if organized. Inputs are 9 numbers for file 1 to file 9. Output is either ascending or mixed.

Sample input:

1,3,2,4,5,6,7,8,9

Sample output:

mixed

10. Write a program.

There are many ways to emphasize my feeling. One of the ways is extending particular letters of a word. For example : so good → sooooooo gooooooooood

To write more o's we can emphasize our feeling. Make a program that repeat the second letter of a word as many time as the input. The first integer input is how many times it will repeat the word. Second input is the word. Output is the extended word.

Sample input:

```
5 bad
```

Sample output:

```
baaaaad
```

Sample input:

```
10 good
```

Sample output:

```
gooooooooooood
```

11. Write a program.

On the Earth, astronauts take a test to go to Mars. It's important because astronauts must be knowledgeable to survive in harsh condition. All the participants take 3 tests and pick the best score. If the participant gets the highest score in a group, he can pass the exam. There are five participants in a group. Make a program that can pick the highest score among them and find who passes the exam. use 2D arrays.

Sample score:

participant 1 : 13, 14, 29
participant 2 : 39, 51, 84
participant 3 : 29, 38, 94
participant 4 : 10, 38, 92
participant 5 : 91, 84, 99

Input : all the scores of each alien from participant 1 to participant 5.

Output : first 5 lines tell the highest score from each alien. Last line tells which alien passed the exam.

Sample input:

13 14 29 39 51 84 29 38 94 10 38 92 91 84 99

Sample output:

the highest score of participant1 is 29
the highest score of participant 2 is 84
the highest score of participant 3 is 94
the highest score of participant 4 is 92
the highest score of participant 5 is 99
participant 5 passed the exam.

A.C.E.

A.C.E.

7. Method

1) What is a Method?

A method is a group of codes that you can reuse. It is helpful for a game developer to express an activity in the real world like run, jump, eat, etc. In other words, a Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

The benefits of using methods are:

- to save time to code the same function;
- to easily find a statement where an error results from; and
- to clearly understand the goal of the codes.

It is important to be able to separate certain part of our codes as methods. We can make any function as a method. In a soccer game program, these activities can be methods.

There are 3 different information in a method, which are **process**, **input**, and **output**. Let's learn them with an example.

Method: baking cookies

```
cookie BakingCookies (flour, water, egg) {
    Knead dough (flour, water, egg)
    Press the cookie cutter.
    Bake in the oven
Return cookie
}
```

- A method should have all the required processes of an activity (or a function).
  Knead, Press, Bake
- A method is able to accept all ingredients (input) for the activity.
  flour,water, egg
- A method can generate output (return value).
  cookie

The syntax of a method is

type methodName (parameters)

```
Ex) cookie c = BakingCookies(2,3,4);      // put the return value to a cookie type variable c.
```

This is the basic form of a method. You have to initialize a method outside of the main method and use it inside the main method. In the method declaration statement above, **type** means the return type of this method's return value, The method will use **parameters** as input values to calculate the **output**.

Example: sumAandB is a method that return the sum of A and B.

```
01    public class learnMethod1 {
02        public static void main(String[] args) {
03            int a = 5, b = 6, c = 7, d = 8;
04            int sum1 = sumAandB(a,b);
05            int sum2 = sumAandB(c,d);
06            System.out.println("a + b = " + sum1);
07            System.out.println("c + d = " + sum2);
08        }
09
10        public static int sumAandB(int a, int b) {
11            return a + b;
12        }
13    }
```

Output:

```
a + b = 11
c + d = 15
```

Note that you have to put "static" in front of the method because main method is a static method. We will study this static quantifier later. When you use a method, be careful with the types of parameters and the type of the return value.

Example: Make a method that returns (a+b)*(a-b). (use float type)

```
public class learnMethod1 {
    public static void main(String[] args) {
        int a = 5, b = 6;
        System.out.println( cal(a,b) );

    }
    public static float cal(float a, float b) {
        return (a+b)*(a-b);
    }
}
```

2) Void type method

If a method has nothing to output, you can use void type.

Example: printACE is a method printing "ACE!" n times.

```
01    public class learnMethod2 {
02        public static void main(String args[]) {
03            int n=5;
```

```
04                  printACE(n);
05          }
06          public static void printACE(int n) {
07                  for(int i=0; i<n; i++) {
08                          System.out.print("ACE!");
09                  }
10          }
11  }
```

Output:

```
ACE!ACE!ACE!ACE!ACE!
```

Example: Make a method that prints times table untill a number n.

Input:

```
n = 3;
```

Output:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
...
3 * 9 = 27
```

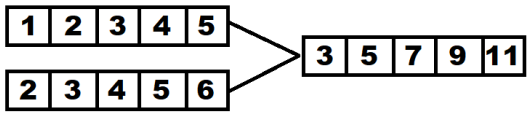

Possible answer:

```
public class learnMethod2 {
        public static void main(String args[]) {
                int n=5;
                timesTable(n);
        }
        public static void timesTable(int n) {
                for(int i=0;i<n;i++) {
                        for(int j=1;j<=9;j++) {
                                System.out.println((i+1) + " * " + j +" = " + (i+1)*j);
                        }
                }
        }
}
```

3) Array type method

You can also set arrays as the type of parameter or the type of return value.

Example: arraySum has two integer arrays and will return an integer array. This method merge two integer arrays to an array. Those arrays have the same length.




```
01  public class learnMethod3 {
02      public static void main(String args[]) {
03          int a[]= {1,2,3,4,5};
04          int b[]= {2,3,4,5,6};
05          int c[]=arraySum(a,b);
06          for(int x: c) {
07              System.out.println(x);
08          }
09      }
10      public static int[] arraySum(int[] a, int[] b) {
11          int c[]= new int[a.length];
12          for(int i=0;i<a.length;i++) {
13              c[i]=a[i]+b[i];
14          }
15          return c;
16      }
17  }
```


Output:

```
3
5
7
9
11
```

Line 05 is to put array a and b as parameters of the method and put the method's return value in array c. In lines 10-16: the method is initialized with two integer array type parameters and an integer array return type.

It is important to note that array c is declared two times: one in main method, the other in arraySum method. A variable declared in a method is called local variable. It will be vanished after finishing the method. So array c in arraySum method is different from array c in main method.

Example: Make a method that find the biggest number in an array and return it .(use int type).

Input:

```
int a[] = {3,5,2,8,4}
```

Output:

```
8
```

4) Methods in a method

Methods can be used inside a method. Let's look at the bakingCookies again.

```
01    cookie BakingCookies(flour, water, egg) {
02          Knead dough(flour, water, egg)
03          Press the cookie cutter.
04          Bake in the oven
05          Return cookie
06    }
```

Lines 02-04 of the BakingCookies are calling another methods. This example initialize these methods and put them together.

Example: BakingCookies method

```
01    public class learnMethod4 {
02          public static void main(String args[]) {
03                int flour = 8, water = 2, egg = 1;
04                String cookieCutter = "ginger bread man";
05
06                String cookie = BakingCookies(flour, water, egg, cookieCutter);
07                System.out.println("We baked " + cookie);
08          }
09
10          public static void kneadDough(int flour, int water, int egg) {
11                System.out.println("kneading dough with flour : " + flour +", water : "+
water +", egg : "+egg);
12          }
13
14          public static void pressCookieCutter(String cookieCutter) {
15                System.out.println("pressing " + cookieCutter + " shaped cookieCutter
on the dough");
16          }
17
18          public static void bakingInOven() {
19                System.out.println("baking oven for 2 hours");
20          }
21
22          public static String BakingCookies(int flour, int water, int egg, String
cookieCutter) {
23                kneadDough(flour, water, egg);
24                pressCookieCutter(cookieCutter);
25                bakingInOven();
26                return cookieCutter + " cookie";
27          }
28    }
```

Output :

```
kneading dough with flour : 8, water : 2, egg : 1
pressing ginger bread man shaped cookieCutter on the dough
baking oven for 2 hours
We baked ginger bread man cookie
```

Example: Make a program about cooking dinner.

This program need to include these three methods.

- Preparing forks and knifes
- Cooking apitigers
- Cooking main dish

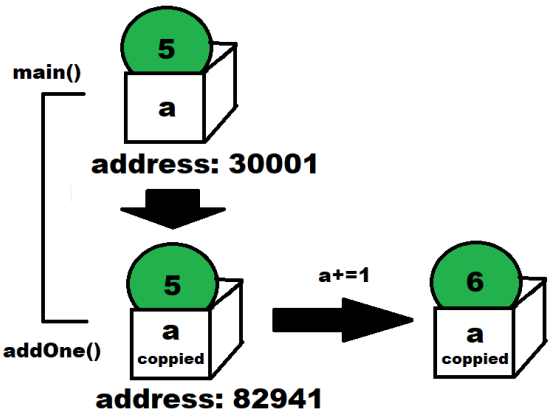
5) Shallow copy and Deep copy

Let's guess the output of the following program:

Example:

```
public class guess1 {
        public static void main(String args[]) {
                int a=5;
                addOne(a);
                System.out.println(a + " from main method");
        }
        public static void addOne(int a) {
                a+=1;
                System.out.println(a + " from addOne method");
        }
}
```

Output:



Why the value of variable a didn't change? It's because shallow copy was performed when calling addOne method.

Inside addOne() method, the parameter "int a" is copied and saved in another memory. Which means the original "a" and the copied "a" are not in the same address. There is nothing these two variables sharing. When copied a changes its value, it's nothing to do with original a.

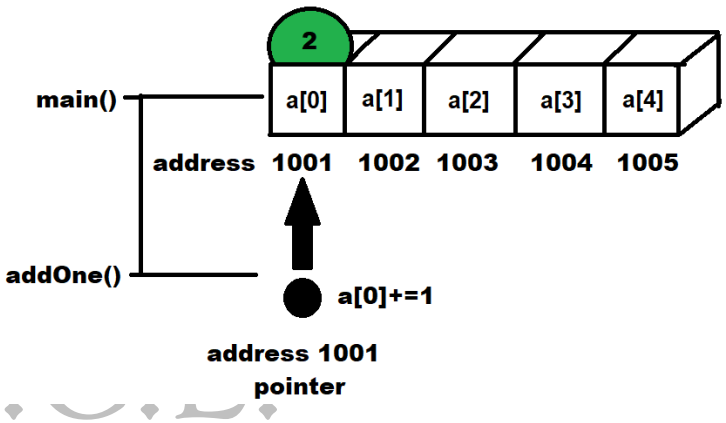
Example: Guess the output of the following program.

```
public class guess2 {
        public static void main(String args[]) {
                int a[]= {1,2,3,4,5};
                addOne(a);
                for(int i=0;i<a.length;i++) {
                System.out.println("a["+i+"] is " + a[i] + " in main method");
                }
        }
        public static void addOne(int[] a) {
                for(int i=0;i<a.length;i++) {
                        a[i]+=1;
                        System.out.println("a["+i+"] is " + a[i] + " in addOne method.");
                }
        }
}
```

Output:

This time, deep copy was performed and the value in main method changed in addOne method? Deep copy doesn't really copy something. It is pointing original variable's address.

It only coppies the address of the variable. Which means it can access to the original value. So if you change the value which is deep coppied, then you will change the original value. If you send an array as a parameter of a method, it will autometically execute the deep copy.



6) Overloading

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

Example: In this program, there are 3 add() methods. But they work a little bit different.

```
01    public class overloading {
02          public static void main(String args[]) {
04                System.out.println(add(10,30));
05                System.out.println(add(10,20,30));
06                System.out.println(add(12.4f,39.41f));
08          }
09
10          public static int add(int a, int b) {
11                return a+b;
12          }
13          public static int add(int a, int b, int c) {
14                return a+b+c;
15          }
16          public static float add(float a, float b) {
17                return a+b;
18          }
19    }
```

Example: Make add() method with two different functions that add all the value of an array and add 2 strings.

7) Homework

1. What does this method do?

```
public static void method1(int a, String b) {
        for(int i=0;i<a;i++) {
                System.out.print(b.charAt(a));
        }
}
```

2. What does this method do?

```
public static void method2(String sArray[]) {
        for(int i=0;i<sArray.length;i++) {
                method1(i,sArray[i]);
        }
}
```


3. What is the problem in this method?

```
public static void method3(int a, int b, int c, int sum) {
        for(int i=0; i<a;i++) {
                for(int j=0; j<b; j++) {
                        for(int k=0; k<c; k++) {
                                sum+=k;
                        }
                }
        }
}
```

4. The isPrime() method find out if a number is prime number or not. Make a program with this method and use these test cases.

Input :	Output:
n = 10	false
n = 13	true
n = 11	true
n = 23	true
n = 48	flase

5. The findPrime() method will find a prime number which is closest and bigger than original number if the number is not a prime number. Use isPrime() method to create findPrime.

Input:	Output:
n = 10	the closest prime number is 11
n = 13	13 is a prime number
n = 96	the closest prime number is 97
n = 23	23 is a prime number
n = 48	the closest prime number is 53

6. AverageScore() will find out the average of the five students' score. Make a program that finds the average of the score.

Input:

Student1 : 95
Student2 : 84
Student3 : 88
Student4 : 92:
Student5 : 99

Output:

Average : 91

7. Sketch a program in the space below that has a void return type method. This method will sort an array in ascending order. Then, use the computer to make and run a program.

8. CHALLENGE: There are three methods. triangleArea(), squareArea(), circleArea(). Each of them find the area of the shape. ACE want to use this method to make a calculator. First input of this calculator is the number of vertexes(if vertex is 0, then exit the calulator,if vertex is 1, then find the area of a circle). Next input depends on the vertexes. For example, if the vertex is 3, then your next input will be base and height. Output is the area of the shape.

Input	Output
Vertex : 3 Base : 10 Height : 2 Vertex: 4 Width: 5 Length : 10 Vertex: 0	Area : 15 Area : 50 EXIT

A.C.E.

8. Class

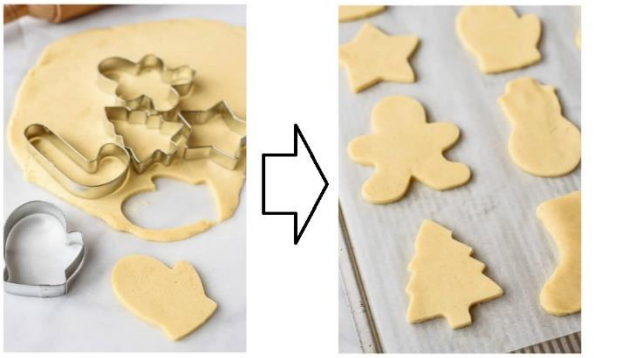
The computer programming is moving to another level, where class is a representative technique of Object Oriented Program. Instead of making one big program, separate it to “objects” and perform each function as necessary. Objects communicate with each other by sending and reciveing data. There are 4 main characteristics in OOP.

Java is an object-oriented programming language. Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake. A Class is like an object constructor, or a "blueprint" for creating objects.

- Encapsulation
- Inheritance
- Polymolphism
- Abstract

In this level of ACE computer science, we are going to learn up to polymolphism.

People are trying to describe, using OOP codes, everything in real life such as human, structures, animal and vehicle. Classes help to describe all these things. A class is a frame of a creature. It contains every characteristic of a creature and it makes blueprint of a creature. Objects will follow the blueprint of a class. So a class only has the consept of a thing just like a cookiecutter.



cookiecutters: class cookies : objects(instance)

For example, if we define a “car” by a class, even though there are many different type of cars, they all follow the rule we built in the class

A programmer is like a God in his/her own program. Now, we are able to make new creatures in a program. First of all, let’s make a creature called “Dog” in our program. What is a dog in the real world? It can be defined as below:

- What are a dog’s features?
  (breed, age, legs, name, tail)
- What can a dog do?
  (bark, run, eat)

1) Three Elements of a Class

i. Fields

A Java field is a variable inside a class. For instance, in a class representing a dog, the Dog class might contain the following fields:

breed, age, legs, name ,tail

ii. Methods

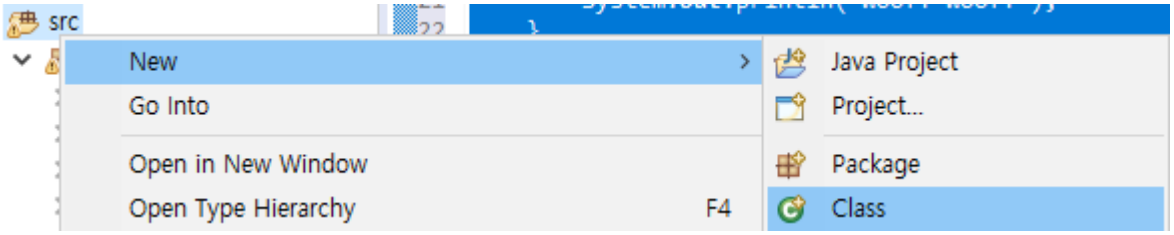
Java methods are where you put the operations on data (variables) in your Java code. In other words, you group Java operations (code) inside Java methods. Java methods must be located inside a Java class. A method is a group of Java statements that perform some operation on some data, and may or may not return a result. For the Dog class, we may have:

bark, run, eat

iii. Constructor

A constructor is a method that has the same name of the class. It always runs one time when you create new object. A constructor is used to initialize the data. In other words, A Java constructor is special method that is called when an object is instantiated. In other words, when you use the new keyword. The constructor initializes the newly created object.

Example: Make a new class and name it as “Dog.”




```
01    public class Dog {
02          //fields
03          int age;
04          int legs;
05          boolean hungry;
06          String breed;
07          String name;
08    }
```

The Dog class only has fields.

We can’t do anything with a class. We need a main class to run a program. So let’s create a new class and name it as “Main.” In Main class, we are going to create two Dogs: “Choco” and “Ginger.”

a) Creating a New Object

This is a code for creating a new object.

ClassName objectName = new ClassConstructor();

Example: Make sure Dog.java and Main.java are in the same project.

```
01  public class Main {
02      public static void main(String arg[]) {
03          Dog choco = new Dog();
04          Dog ginger = new Dog();
05
06          choco.name= "Choco";
07          ginger.name = "Gineger";
08
09          System.out.println(choco.name);
10          System.out.println(ginger.name);
11      }
12  }
```

Even though there is no constructor, the compiler create a default constructor which has no parameter. Without parameter, we need to initialize every variable in the field. Let's find out features that are common to all dogs. (They have 4 legs and they are always hungry.)

Example: Update your Dog.java

```
01  public class Dog {
02      //fields
03      int age;
04      int legs;
05      boolean hungry;
06      String breed;
07      String name;
08
09      //constructor
10      Dog(int age, String name, String breed){
11          this.age = age;
12          this.name = name;
13          this.breed = breed;
14          legs=4;
15          hungry=true;
16      }
17  }
```

If the name of a parameter and the name of a field are the same, compiler will get confused which data to call. The keyword "**this**" let you access to the class where you are working in. So, "this.age" is the same as "Dog.age."

b) Constructor

By using a constructor, we can initialize fields easily.

Example: Update your Main.java

```
01  public class Main {
02      public static void main(String arg[]) {
03          Dog choco = new Dog(3,"Choco","Yorkshire Terrier");
04          Dog ginger = new Dog(5, "Ginger","Golden Retriever");
05
07          System.out.println(choco.name);
08          System.out.println(choco.breed);
09          System.out.println(ginger.age);
10          System.out.println(ginger.hungry);
11      }
12  }
```

Creating and initializing an object at the same time by a constructor. We can also have multiple constructors with different parameters.

c) Method

Now, we are going to add methods. What where the activities a dog do?

Example: Update your Dog.java


```
01  public class Dog {
02      //fields
03      int age;
04      int legs;
05      boolean hungry;
06      String breed;
07      String name;
08
09      //constructor
10      Dog(int age, String name, String breed){
11          this.age = age;
12          this.name = name;
13          this.breed = breed;
14          legs=4;
15          hungry=true;
16      }
17
18      //methods
19      void bark() {
20          System.out.println("Woof! Woof!");
21      }
22      void run() {
23          System.out.println(name + "is running very fast!");
```

```
24              }
25              void eat() {
26                      if(hungry) {
27                              System.out.println(name + "is eating.");
28                              hungry = false;
29                      }
30                      else {
31                              System.out.println(name + "is full.");
32                      }
33              }
34      }
```

We are ready to use the class defined above to complete a program.

```
01      public class Main {
02              public static void main(String arg[]) {
03                      Dog choco = new Dog(3,"Choco","Yorkshire Terrier");
04                      Dog ginger = new Dog(5, "Ginger","Golden Retriever");
05
06                      System.out.println(choco.name + " has " + choco.legs + " legs.");
07                      System.out.println(ginger.name + " has " + ginger.legs + " legs ");
08                      System.out.println(choco.name + " is " + choco.age + " years old");
09
10                      choco.bark();
11                      ginger.bark();
12                      choco.eat();
13                      ginger.run();
14              }
15      }
```


Example: Make a Cat class.

Conditions: at least 3 fields
at least 3 methods
at least 2 constructors
create 3 different cats
create Cat.java and Main2.java

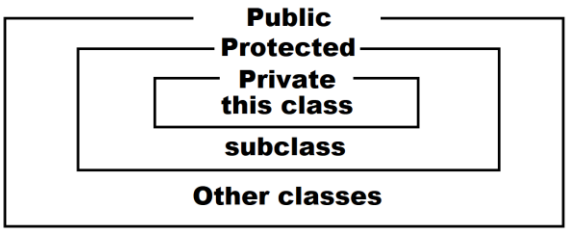
Possible solution: Cat.java

```
public class Cat {
        String name;
        String eyeColor;
        int box;

        Cat(String name){
                this.name = name;
                eyeColor = “Green”;
                box = 0;
        }

        Cat(String name, String eyeColor){
                this.name = name;
                this.eyeColor = eyeColor;
                box = 0;
        }

        void scratch() {
                System.out.println(name + " scratched me!!!");
        }

        void takeBox() {
                System.out.println(name + " took my box!!");
                box+=1;
        }

        void jump() {
                System.out.println(name + "!! Stop jumping!!");
        }
}

public class Main2 {
        public static void main(String args[]) {
                Cat kitty = new Cat("Kitty","Odd Eye");
                Cat nami = new Cat("Nami");

                kitty.takeBox();
                kitty.jump();
                nami.scratch();
        }
}
```

2) Access modifiers

To make a program, many classes are communicating. What if a class trys to modify an important data in other class? We have to stop it. Access modifiers hide the important data from other classes. Three levels of access modifier exsist:

- public: expose to all classes (default)
- protected: exposes to subclass only
- private: hide from other classes. (can only be accessed from "this" class)



A Java access modifier specifies which classes can access a given class and its fields, constructors and methods. Access modifiers can be specified separately for a class, its constructors, fields and methods. Java access modifiers are also sometimes referred to in daily speech as *Java access specifiers*, but the correct name is Java access modifiers. Classes, fields, constructors and methods can have one of three different Java access modifiers:

We are going to talk about "protected" after we learn inheritance. For now, let's identify the differences between public and private.

Example: Access Modifiers

```
01  public class AccessModifier {
02      public int lessImportantData;
03      private int veryImportantData;
04          …
05  }
06
07  public class Main3 {
08      public static void main(String[] args) {
09          AccessModifier am = new AccessModifier();
10          am.lessImportantData = 10;
11          am.veryImportantData = 10;
12      }
13  }
```

There is an error in line 11 of Main.java. Because veryImportantData is a private variable, it can't be accessed from Main3 class.

Usually, people set all the fields to private. Then how can other class change the value? For this purpose, we use two methods: getter and setter.

- getter: return a field variable.
- setter: set a value at a field.

Getter and setter are methods that has particular function. Other classes can access to the field by calling these methods.

Example: In this example, AccessModifier2 has getter and setter.

```
01      public class AccessModifier2 {
02              private int field1;
03              private String field2;
04
05              int getField1(){
06                      return field1;
07              }
08              String getField2() {
09                      return field2;
10              }
11
12              void setField1(int field1) {
13                      this.field1 = field1;
14              }
15
16              void setField2(String field2) {
17                      this.field2 = field2;
18              }
19      }
21
21      public class Main4 {
22              public static void main(String args[]) {
23                      AccessModifier2 am2 = new AccessModifier2();
24
25                      am2.setField1(10);
26                      am2.setField2("Hi modifier");
27
28                      System.out.println(am2.getField1());
29                      System.out.println(am2.getField2());
30              }
31      }
```


Output:

```
10
Hi modifier
```

Example: Add “private” access modifier to the fields of Cat and Dog classes and add getter and setter. Change the main class in order to make it work.

Possible answer:

```
// Dog.java
public class Dog {
        //fields
        private int age;
        private int legs;
        private boolean hungry;
        private String breed;
        private String name;

        //constructor
        Dog (int age, String name, String breed) {
                this.age = age;
                this.name = name;
                this.breed = breed;
                legs=4;
                hungry=true;
        }

        //getter and setter
        int getAge() {
                return age;
        }
        int getLegs() {
                return legs;
        }
        boolean getHungry() {
                return hungry;
        }
        String getBreed() {
                return breed;
        }
        String getName() {
                return name;
        }

        void setAge(int age) {
                this.age = age;
        }
        void setLegs(int legs) {
                this.legs = legs;
        }
        void setHungry(boolean hungry) {
```

```
            this.hungry = hungry;
        }
        void setBreed(String breed) {
            this.breed = breed;
        }
        void setName(String name) {
            this.name = name;
        }

        //methods
        void bark() {
            System.out.println("Woof! Woof!");
        }
        void run() {
            System.out.println(name + "is running very fast!");
        }
        void eat() {
            if(hungry) {
                System.out.println(name + "is eating");
                hungry = false;
            }
            else {
                System.out.println(name + "is full");
            }
        }
}

// Main.java
public class Main {
    public static void main(String arg[]) {
        Dog choco = new Dog(3,"Choco","Yorkshire Terrier");
        Dog ginger = new Dog(5, "Ginger","Golden Retriever");

        System.out.println(choco.getName() + " is " + choco.getAge() + " years old.");
        ginger.eat();
    }
}

// Cat.java
public class Cat {
    //fields
    private String name;
    private String eyeColor;
    private int box;
```

```
        //constructor
        Cat(String name){
                this.name = name;
                eyeColor = "Green";
                box = 0;
        }
        Cat(String name, String eyeColor){
                this.name = name;
                this.eyeColor = eyeColor;
                box = 0;
        }

        //getter and setter
        String getName() {
                return name;
        }
        String getEyeColor() {
                return eyeColor;
        }
        int getBox() {
                return box;
        }

        void setName(String name) {
                this.name = name;
        }
        void setEyeColor(String eyeColor) {
                this.eyeColor = eyeColor;
        }
        void setBox(int box) {
                this.box= box;
        }

        //methods
        void scratch() {
                System.out.println(name + " scratched me!!!");
        }

        void takeBox() {
                System.out.println(name + " took my box!!");
                box+=1;
        }
        void jump() {
                System.out.println(name + "!! Stop jumping!!");
        }
}
```

```
// Main2.java
public class Main2 {
	public static void main(String args[]) {
		Cat kitty = new Cat("Kitty","Odd Eye");
		Cat nami = new Cat("Nami");

		nami.setEyeColor("White");

		kitty.takeBox();
		kitty.takeBox();
		System.out.println(kitty.getName()+" now has " + kitty.getBox() + " boxes");
		kitty.jump();
		nami.scratch();
	}
}
```

3) The keyword "static"

Even though we don't know the meaning of static, the keyword "static" has been used many times since we wrote main method. There are two different place where we use the keyword "static"

- -static variable: It is a class level variable common to all instances (objects) of the class. This variable should be called as a class, not only as object.
- -static method : Java static methods belong to the class. They use no instance variable and usually take input from parameters, perform actions, then return some results.

Example: Update Dog class.

```
01  public class Dog {
02      //fields
03      private static int dogCount=0;      //static variable
04      private int age;
05      private int legs;
06      private boolean hungry;
07      private String breed;
08      private String name;
09
10      //constructor
11      Dog(int age, String name, String breed) {
12          this.age = age;
13          this.name = name;
14          this.breed = breed;
15          legs=4;
16          hungry=true;
17          dogCount++;
18      }
19
20      //getter and setter
21      int getAge() {
22          return age;
23      }
24      int getLegs() {
25          return legs;
26      }
27      boolean getHungry() {
28          return hungry;
29      }
30      String getBreed() {
31          return breed;
32      }
33      String getName() {
34          return name;
35      }
```

```
36
37              void setAge(int age) {
38                      this.age = age;
39              }
40              void setLegs(int legs) {
41                      this.legs = legs;
42              }
43              void setHungry(boolean hungry) {
44                      this.hungry = hungry;
45              }
46              void setBreed(String breed) {
47                      this.breed = breed;
48              }
49              void setName(String name) {
50                      this.name = name;
51              }
52
53              //methods
54              static void howManyDogs() {
55                      System.out.println("You have " + dogCount + "dogs");
56              }       //static method
57              void bark() {
58                      System.out.println("Woof! Woof!");
59              }
60              void run() {
61                      System.out.println(name + "is running very fast!");
62              }
63              void eat() {
64                      if(hungry) {
65                              System.out.println(name + "is eating");
66                              hungry = false;
67                      }
68                      else {
69                              System.out.println(name + "is full");
70                      }
71              }
72      }
```


Update Main class

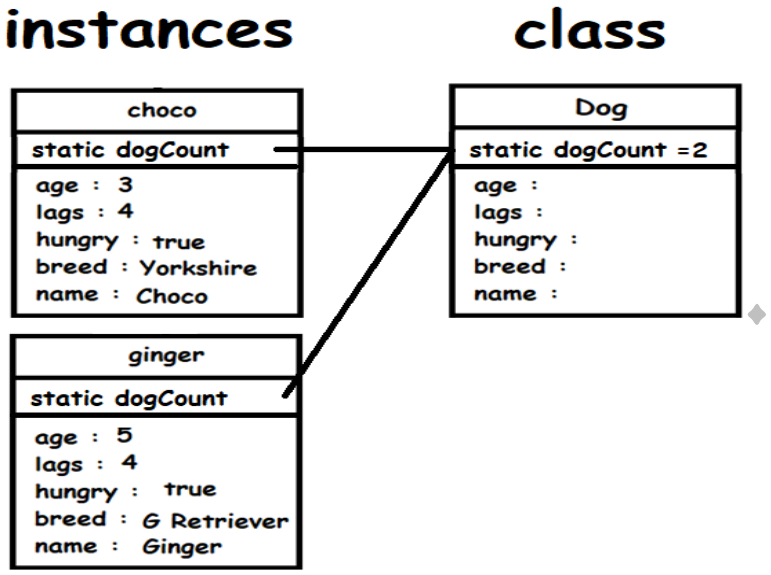
```
01      public class Main {
02              public static void main(String arg[]) {
03                      Dog choco = new Dog(3,"Choco","Yorkshire Terrier");
04                      Dog ginger = new Dog(5, "Ginger","Golden Retriever");
05
06                      System.out.println(choco.getName());
07                      System.out.println(ginger.getName());
```

```
08
09                  choco.howManyDogs();
10                  ginger.howManyDogs();
11                  Dog.howManyDogs();
14              }
15      }
```

Guess the output first and then run the code.

Output:

Choco
Ginger
You have 2dogs
You have 2dogs
You have 2dogs



4) Practice

1. Find errors if any.

ACEManagement.java

```
01  public class ACEManagement {
02      String studentName;
03      int studentGrade;
04
05      ACEManagement(String studentName, int studentGrade){
06          studentName = studentName;
07          studentGrade = studentGrade;
08      }
09  }
```

Main5.java

```
01  public class Main5 {
02      ACEManagement am = new ACEManagement("Ryan");
03      System.out.println(am.studentName);
04  }
```

Answer:

2. Find errors if any.

ACEManagement.java

```
01  public class ACEManagement {
02      private String teacherName;
03      private String subject;
04  }
```

Main5.java

```
01  public class Main5 {
02      ACEManagement am = new ACEManagement("Sam","Director");
03      am.teacherName = "Aaron";
04      am.subject = "Computer Science"
05  }
```

Answer:

3. Fix errors if any and then guess the output.

ACEManagement.java

```
01  public class ACEManagement {
02      private static int teacherCount = 0;
03      String teacherName;
04      String subject;
05
06      ACEManagement(){
07          teacherCount++;
08      }
09
10      static int countTeacher() {
11          return teacherCount;
12      }
13  }
```

Main5.java

```
01  public class Main5 {
02      public static void main(String args[]) {
03          ACEManagement aaron = new ACEManagement();
04          int a = aaron.countTeacher();
05          ACEManagement thomas = new ACEManagement();
06          int b = thomas.countTeacher();
07          int c = ACEManagement.countTeacher();
08
09          System.out.println(a+b+c);
10      }
11  }
```


Answer:

Output :

4. Account class handles a person's money. People can deposit by calling deposit() method, withdraw by calling withdraw() method, and check balance by calling check() method. Make an account program. All fields have to be private.

Console activities:

```
Hello. What do you want to do?(1.deopsit, 2.withdraw, 3.check balance, 4.exit): 1
How much do you want to deposit? : 3000
Deposit Succeed.
Hello. What do you want to do?(1.deopsit, 2.withdraw, 3.check balance, 4.exit): 2
How much do you want to withdraw? : 1500
Withdraw Succeed.
Hello. What do you want to do?(1.deopsit, 2.withdraw, 3.check balance, 4.exit): 3
Your available balance is 1500.0
Hello. What do you want to do?(1.deopsit, 2.withdraw, 3.check balance, 4.exit): 4
Thank you! Have a nice day!
```

A.C.E.

5. ACE is making a program that handle's all ACSL score information. ACEScore class handles name, grade and test score. Especially, this program also finds the best score and the name of the student. In main method, create 5 instances and find the highest score and name by using static variable.

6. Challenge: Sam has a lot of consult meeting everyday. Sometimes people come to consult while other consulting is in progress. Aaron decided to make a consult reservation program. Sam's working hour is 1pm to 9pm. In this program, people can make appointments in every hour sharp from 1pm to 8pm. If there is an appointment, that person should choose another time. In addition, sam can check how many appointments he has and he can check the time of appointments.

Console activities:

Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 1
Please write your name : aaron
Please choose the time 1 - 8 pm : 3
Successfully Booked! Thank you!
Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 1
Please write your name : thomas
Please choose the time 1 - 8 pm : 1
Successfully Booked! Thank you!
Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 1
Please write your name : sam
Please choose the time 1 - 8 pm : 3
It's already booked. Please choose the other time.
Please choose the time 1 - 8 pm : 4
Successfully Booked! Thank you!
Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 2
you have 3 appointments today
Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 3
Here are appointments today.
1am : thomas
3am : aaron
4am : sam
Welcome to ACE. what are you looking for?(1.make a reservation 2.check appointment numbers 3.check appointment time 4.EXIT): 4
Thank you! Have a nice day! EXIT

A.C.E.

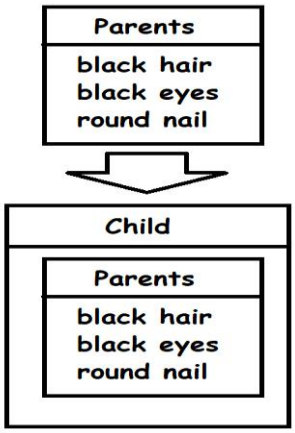
A.C.E.

A.C.E.

9. Inheritance

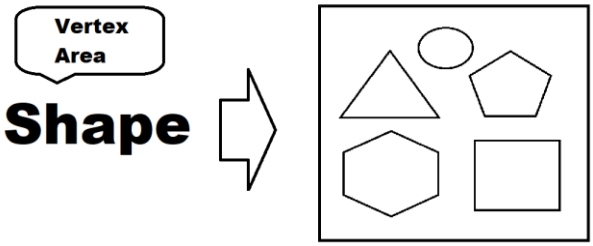
Class was like a capsule that has information about a thing. Now we are going to talk about second characteristic of OOP, inheritance. Normally, people inherit property from their parents. Also, people inherit genetic information from their parents, too. That means we have some features inherited from our parents.

This logic can be described in a class, too. There are parents class (Super class) and child class (Sub class)



1) Super class and Sub class

Super class gives features to sub class. The keyword "extends" connects super class and sub class. Let's talk about an example of super class.



- What is a shape?
- What are the features of the shape?

All shapes has vertex and area. Let's make Shape class as a super class.

Example: Shape class

```
01    public class Shape {
02        int vertex;
03
04        void setVertex(int vertex) {
05            this.vertex = vertex;
06        }
07
08        int getVertex() {
09            return vertex;
10        }
11
12        void area() {
13            System.out.println("A shape has an area");
14        }
15    }
```

What can be a child of a shape?

- Can a rectangle be a child of a shape?
- Does a rectangle have all features a shape has?

- How many vertexes does a rectangle have?
- How can we calculate the area of a rectangle?
- What other features does a rectangle have?

Example:

```
// Rectangle class
// Focus on the keyword "extends." We can access all features that Shape class has.
01 public class Rectangle extends Shape {
02      private double length;
03      private double width;
04
05      void setLength(double length) {
06              this.length = length;
07      }
08      double getLength() {
09              return length;
10      }
11
12      void setWidth(double width) {
13              this.width = width;
14      }
15      double getWidth() {
16              return width;
17      }
18
19      Rectangle(double length, double width){
20              this.setVertex(4);
21              this.length = length;
22              this.width = width;
23      }
24      Rectangle(){
25              this.setVertex(4);
26      }
27
28      void area() {
29              System.out.println("The area of this rectangle is " + length * width);
30      }
31 }

// main class
public class main {
        public static void main(String args[]) {
                Rectangle rect1 = new Rectangle(3,4);
                rect1.area();
                System.out.println("A rectangle has " + rect1.vertex + " vertexes.");
        }
}
```

Output:

```
The area of this rectangle is 12.0
A rectangle has 4 vertexes.
```

Example: Make Circle and Triangle classes. Think about the features of each class and create. Let's say a circle has only one vertax.

Possible answer:

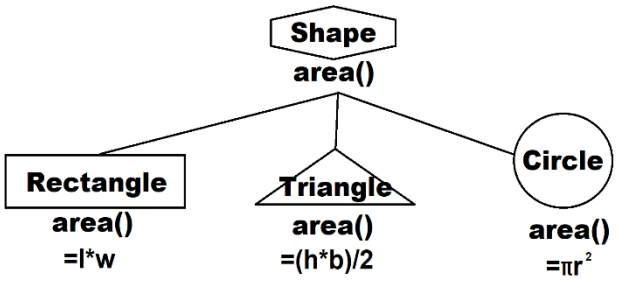
```
// Circle.java
public class Circle extends Shape{
	private double redius;

	private double getRedius() {
		return redius;
	}
	void setRedius(double redius) {
		this.redius = redius;
	}

	Circle(double redius){
		this.redius=redius;
		vertex= 1;
	}

	void area() {
		System.out.println("The area of this circle is : " + redius*redius*Math.PI);
	}
}

// Triangle.java
public class Triangle extends Shape{
	private double base;
	private double height;

	double getBase() {
		return base;
	}
	double getHeight() {
		return height;
	}

	void setBase(double base) {
		this.base=base;
	}
	void setHeight(double height) {
```

```
            this.height=height;
        }

        Triangle(double base, double height){
            vertex = 3;
            this.base = base;
            this.height = height;
        }

        void area() {
            System.out.println("The area of this triangle is " + base*height/2);
        }
    }
```

2) Overriding

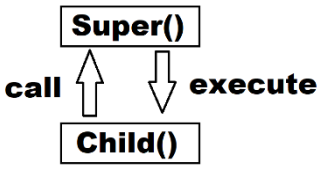
There are area() methods in Shape class, Rectangle class, Circle class, and Triangle class. How can it be decided which method to call? This is called "Overriding".

Overriding means that child classes can have the methods under the same name of the methods in super class. The newly declared method functions differently. If the named method is called, the method in their own class will be called. It shows the "Polymorphysm" characteristic of OOP.



Constructor

When we call the constructor from a child class, the constructor of the super class should always be called. In other word, super class's constructor is inside the child class's constructor as a default.



If super class's constructor has parameters, it has to set all the parameters in child class's constructor. In this situation, we use "super" keyword.

Do you remember "this" keyword? Similarly, the keyword "**super**" allows access to eliments in super class.

- super.vertex access to a variable in super class
- super.area() access to a method in super class
- super(a) access to a constructor in super class

Example:

```
// super class
01      public class SuperClass {
02              int v1;
03              SuperClass(int v1){
04                      this.v1=v1;
05              }
06      }

// child class
01      public class ChildClass extends SuperClass{
02              ChildClass(int v1){
03                      super(v1);      //Constructor of the super class
04              }
05      }
```

3) Practice

1. Find errors if any.

```
// Super1.java
01      public class Super1 {
02              int a;
03              int b;
04
05              Super1(int a, int b){
06                      this.a = a;
07                      this.b = b;
08              }
09      }

// Child1.java
01      public class Child1 extends Super1{
02              float c;
03              float d;
04
05              Child(float c, float d){
06                      super();
07                      this.c=c;
08                      this.d=d;
09              }
10      }
```


Answer :

2. Find errors if any.

```
// Super2.java
01      public class Super2 {
02              String s1;
03              String s2;
04
05              Super2(int s1, int s2){
06                      this.s1=s1;
07                      this.s2=s2;
08              }
09
10              String methodSuper() {
11                      return s1+s2;
12              }
13      }

// Child2.java
01      public class Child2 extends Super2 {
02              String s3;
03              String s4;
04
05              Child2(String s1, String s2, String s3, String s4){
06                      super(s1,s2);
07                      this.s3=s3;
08                      this.s4=s4;
09              }
10              String methodSuper() {
11                      return s1+s3;
12              }
13      }
```


Answer:

3. There are many different cars in the world. If we separate them by category, the categories are car, motorcycle and bus. Make a Vehicle super class and make subclasses for car, motocycle and bus, respectively.

Hint:

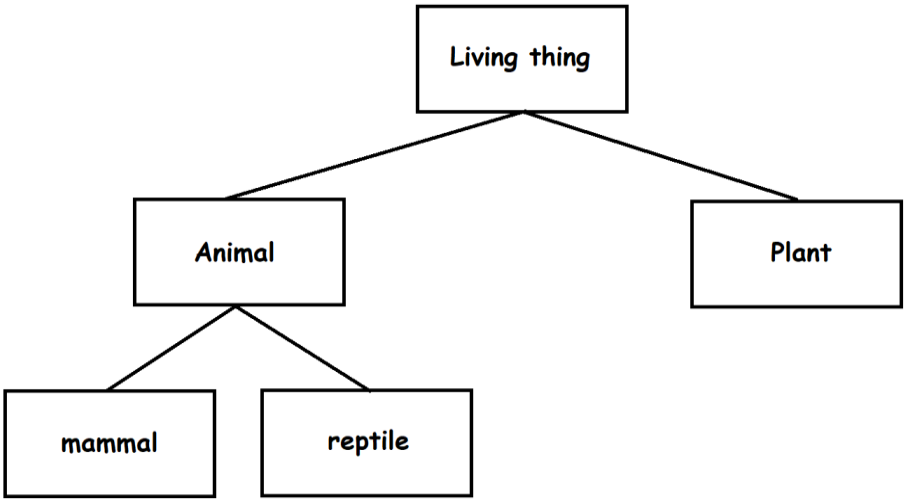
What are their common characteristics?
What are the differences?
At least use 2 fields can be defined: one method in super class and another in child class.

4. We live with many devices: mobile phones, computers, printers and so on. Make Device super class and make three specific child devices.

Hint:

At least use 2 fields: one method in super class and another in child class.

5. Challenge: What is a human being? A human is a living thing, an animal, and a mammal. Make a class structures like the picture below. And make an object of yourself which has a mammal's characteristic.



A.C.E.

A.C.E.

10. Practical Programming (1/3)

1) Project COIN STRIP

PROBLEM: The following game comes from the book "On Numbers and Games" by John Conway. Given the rules of the game Coin Strip, play a next move. The rules are as follows:

a) The board consists of a series of adjacent squares. The number of squares is variable.

b) At the start of the game, coins (markers) are placed on the board by one of the two players.

c) Coins can't share the same square.

d) Coins can't pass another coin.

e) Coins can only move to the left.

f) Coins may move any number of squares using the rules above.

g) The winner is the last player who moves a coin.

A sample initial game board set up is shown below:

INPUT: There will be 5 input lines. Each line will include the number of squares on the board, the number of coins on the board and the location numbers where coins are placed. In the example above there are 12 squares and 7 coins. The coins are at locations 1, 4, 6, 8, 9, 10 and 12.

OUTPUT: For each input line, print the number of coins that can be moved as a starting move 1, 2, 3, 4 or 5 squares. For the board above the coin in location 4 could be moved 1 or 2 squares. The coin in location 6 can only be moved 1 square. The coin in location 8 can only be moved 1 square. The coin in location 12 can only be moved 1 square. The output would be 4,1,0,0,0. That is 4 coins could be moved 1 square to the left, 1 coin could be moved 2 squares to the left but no coins could be moved 3, 4 or 5 squares to the left.

SAMPLE INPUT	SAMPLE OUTPUT
1. 12, 7, 1, 4, 6, 8, 9, 10, 12	1. 4,1,0,0,0
2. 14, 6, 1, 3, 6, 7, 10, 14	2. 4,3,1,0,0
3. 10, 2, 4, 9	3. 2,2,2,1,0
4. 10, 2, 6, 10	4. 2,2,2,1,1

Hint:

i. See the sample gameboard. How can you describe this data structure in codes?
ii. See the sample inputs. How can you separate useful values and useless values?
iii. Draw a blueprint of your algorithm.
iv. Find patterns that make your code efficient.
v. With the patterns, build new blueprint.
vi. Program it with the blueprint.

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.

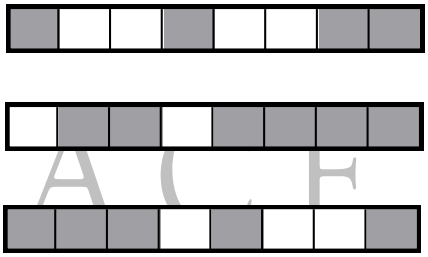
2) Project The Next Generation

PROBLEM: Cellular Automata is a branch of mathematics that studies the effect of recursively applying a set of rules to a beginning state of cells in a grid. This area of study was made famous by John Horton Conway's publication in 1970 of the Game of Life. In this program the grid will be one-dimensional. The beginning state of each cell as described by Conway is alive or dead. In the grid below, live cells are shaded and dead cells are white. Once the rules are applied to all the cells, a new grid is produced and is called the next generation. The beginning grid, its next two generations and the sample rules for the example given below are:

i. A cell will be alive in the next generation if a) the neighbor cell to its left is alive and the neighbor cell to its right is dead or b) the neighbor cell to its left is dead and the neighbor cell to its right is alive.

ii. A cell will be dead in the next generation if a) the neighbor cell to its left is alive and the neighbor cell to its right is alive or b) the neighbor cell to its left is dead and the neighbor cell to its right is dead.

iii. If there is no neighbor cell to the left or right of a cell, then consider those nonexistent cells to be dead.



INPUT: There will be 5 lines of input. Each line will give the number of cells in the grid, the state of each cell in A or D format and the generation to output. For the example above the input is given on line #1.

OUTPUT: For each input line, print the state of the grid for the generation number given.

SAMPLE INPUT	SAMPLE OUTPUT
1. 8, A, D, D, A, D, D, A, A, 2	1. AAADADDA
2. 4, A, A, A, A, 2	2. DAAD
3. 5, A, D, D, A, D, 3	3. ADAAD
4. 6, A, A, D, D, A, D, 4	4. AAADDA
5. 8, D, A, A, D, D, A, D, D, 5	5. DADDAADA

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.

11. Practical Programming (2/3)

1) Project Digital Deletions

PROBLEM: Given a sequence of digits, modify the sequence by:

a) Deleting all zeros, if any, and all the digits to their left

b) Changing a digit to a value less than that digit (by some set of rules). No negatives allowed.

In a real game the player who removes the last digit loses.

For this program, given the sequence, remove all the zeros, if any, and all the digits to their left. Then find the largest remaining digit and if it is even subtract 2 from it or if it is odd subtract 1 from it. If two or more digits become tied as the largest digit use the rightmost digit as the largest. Repeat the application of the rules to the sequence. How many moves were required to delete the sequence? Two examples are shown below:

MOVES	3 5 8 0 2 5 4	8 3 0 7 5 1
1	2 5 4	7 5 1
2	2 4 4	6 5 1
3	2 4 2	4 5 1
4	2 2 2	4 4 1
5	2 2 0	4 2 1
6	EMPTY	2 2 1
7		2 0 1
8		1
9		0
10		EMPTY

INPUT: There will be 5 lines of input. Each input will consist of a sequence of positive integers. The first integer will tell how many integers are in the sequence that will follow.

OUTPUT: Print the number of moves required to delete all the digits.

SAMPLE INPUT
1. 7, 3, 5, 8, 0, 2, 5, 4
2. 6, 8, 3, 0, 7, 5, 1
3. 5, 5, 0, 6, 0, 4

SAMPLE OUTPUT
1. 6
2. 10
3. 4

Hint:

i. What are conditions of this problem?
ii. Draw a cycle of process
iii. Separate the process by 3.
iv. Name these three process and make methods with each of them.
v. Make a method “solve()” which has three methods of Step4.
vi. Program it.

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.

2) Project Pinochle

PROBLEM: Pinochle uses a deck of 48 cards that is dealt out to the players (16 cards each). A deck has 2 of each type of card (Ace, King, Queen, Jack, 10, 9) per suit (diamonds, clubs, spades, hearts). The first step in playing the game is to count the number of points in a hand using the following rules:

	CASE	POINTS		CASE	POINTS
1	2 Jacks of Diamonds & 2 Queens of Spades	30	6	1 Jack of Diamonds & 1 Queen of Spades	4
2	4 Aces (1 in each suit)	10	7	4 Queens (1 in each suit)	6
3	4 Kings (1 in each suit)	8	8	4 Jacks (1 in each suit)	4
4	8 Aces (2 in each suit)	100	9	8 Queens (2 in each suit)	60
5	8 Kings (2 in each suit)	80	10	8 Jacks (2 in each suit)	40

The same card can be used in multiple cases but only once in each case. If a hand has two Jacks of diamonds and two Queens of spades, those cards could be used in Case #1 and Case #6 to accumulate 38 points.

INPUT: There will be 5 input lines each representing the cards in one hand. Each input line will contain 4 strings. The strings name the cards in each suit in that hand. The order of the strings will always be diamonds, clubs, spades and hearts. The card names used will be A (Ace), K (King), Q (Queen), J (Jack), T (Ten) and N (Nine). There will always be at least one card in each suit.

OUTPUT: The score for each input hand.

SAMPLE INPUT

1. ATKQQJ, AKQQ, KQQJN, A
2. KQN, ATTQN, AQJ, ATKQJ
3. AAKQJNN, TN, TTKNN, KQ
4. TKQJ, ATKQJ, AAKQJ, JN

SAMPLE OUTPUT

1. 4
2. 6
3. 0
4. 8

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.

12. Practical Programming (3/3)

1) Project Sudoku

PROBLEM: The 9×9 grid below is a Sudoku puzzle. Sudoku puzzles are all the rage in the newspapers in London, England and now in US cities. This puzzle is a sample from the www.sudoku.com web site. The object of the puzzle is to place the digits 1 – 9 in the puzzle such that each digit appears just once in each row and each column. Further, each digit can only appear once in each of the bounded 3×3 grids. In this program you will not be asked to solve the puzzle, but will be asked to list the possible digits for a given cell of the grid.

	5			1			4	
1		7				6		2
			9		5			
2		8		3		5		1
	4			7			2	
9		1		8		4		6
			4		1			
3		4				7		9
	2			6			1	

INPUT: There will be 14 lines of input. Each of the first 9 lines will represent one row of the grid. Blanks will be represented by 0's. The first line of the grid above would be represented by: 0,5,0,0,1,0,0,4,0. The first 0 is at location (1,1) and the adjacent 5 is at location (1,2). Although the digits are shown separated by commas, the first 9 lines can be entered in any manner. The last 5 lines will each contain 2 digits representing a row and column number.

OUTPUT: For each row and column number, print all the possible digits that could occupy that grid location. The digits can be printed in any order.

SAMPLE INPUT

1. 0,5,0,0,1,0,0,4,0
2. 1,0,7,0,0,0,6,0,2
3. 0,0,0,9,0,5,0,0,0
4. 2,0,8,0,3,0,5,0,1
5. 0,4,0,0,7,0,0,2,0
6. 9,0,1,0,8,0,4,0,6
7. 0,0,0,4,0,1,0,0,0
8. 3,0,4,0,0,0,7,0,9
9. 0,2,0,0,6,0,0,1,0
10. 4,2
11. 4,8
12. 6,8
13. 8,8
14. 3,5

SAMPLE OUTPUT

1. 6,7
2. 7,9
3. 3,7
4. 5,6,8
5. 2,4

Hint:

i. How can you express this Sudoku Board in codes?
ii. Wirte 3 main rules of Sudoku.
iii. Blueprint 3 main rules of Sudoku as methods.
iv. Make a method “solve()” which contains three methods of Step iii.
v. Program it.

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.

2) Project Reversi

PROBLEM: According to Wikipedia, Reversi (also known as Othello) is an abstract strategy board game which involves play by two parties on an eight-by-eight square grid with pieces that have two distinct sides. Pieces have an X on one side and an O on the other side, each side representing one player. The object of the game is to make your pieces constitute a majority of the pieces on the board at the end of the game, by flipping as many of your opponent's pieces as possible. Our example starts with the pieces placed as in the figure in upper left.

		A	B	C	D	E	F	G	H
8									
7									
6									
5					X	O			
4					O	X			
3									

	A	B	C	D	E	F	G	H
8								
7								
6					#			
5				X	O	#		
4			#	O	X			
3				#				

	A	B	C	D	E	F	G	H
8								
7								
6					X			
5				X	X			
4				O	X			
3								

	A	B	C	D	E	F	G	H
8								
7								
6				#	X	#		
5				X	X			
4				O	X	#		
3								

X goes first. X tries to place a piece with the X side up on the board next to an O, in such a position that there exists at least one straight (horizontal, vertical, or diagonal) line between the new piece and another X piece, with one or more contiguous (touching at some boundary line or point) O pieces between them. In the figure in the upper right, X has the options indicated by #'s:

After placing the piece, X flips over all O pieces lying on a straight line between the new piece and any anchoring X pieces. All flipped pieces now show the X side, and X can use them in later moves -- unless O has flipped them back in the meantime.

If X decided to place the piece at location 6E, the result would be as in the figure on the lower left. Now, its O's turn to play. O could put a piece at any of the locations indicated with a # as shown in the figure on the lower right to flip an X.

INPUT: There will be 5 input lines. Each line will give the position of the pieces in a game. Each line will give the number of X's followed by their 2-character locations in row-column order. This will be followed by the number of O's followed by their 2-character locations in row-column order.

OUTPUT: For each line of input, print (in any order) all the locations where an X could be placed such that at least one O piece can be flipped. If an X can't be so placed, print NONE.

SAMPLE INPUT

1. 2, 5D, 4E, 2, 5E, 4D
2. 4, 6E, 5D, 5E, 4E, 1, 4D
3. 3, 5D, 5E, 6E, 3, 4D, 4E, 4F
4. 3, 3E, 4E, 6E, 5, 4D, 4F, 5D, 5E, 6D
5. 5, 6C, 6D, 6E, 4E, 3E, 5, 4D, 4F, 5C, 5D, 5E

SAMPLE OUTPUT

1. 6E, 5F, 4C, 3D
2. 4C, 3C, 3D
3. 3C, 3D, 3E, 3F, 3G
4. 6C, 5C, 4C, 4G, 5G
5. 6B, 4B, 4C, 3D, 3G, 4G, 5G

Sketch your blueprint below

Use the space below to organize your idea before typing the keyboard.

A.C.E.