

Lecture 1

What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than 3 billion devices run Java.

It is used for:

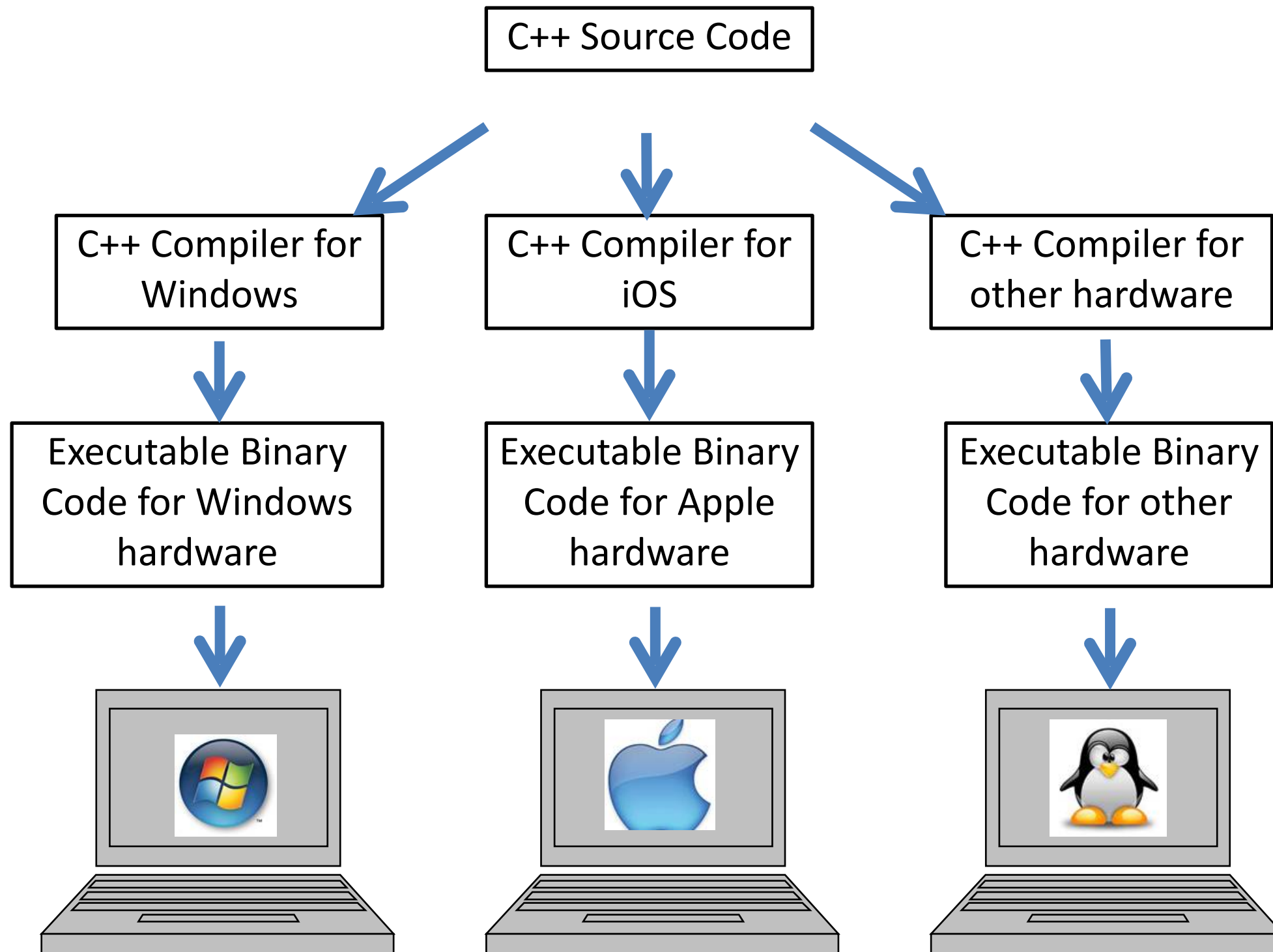
- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

Why Use Java?

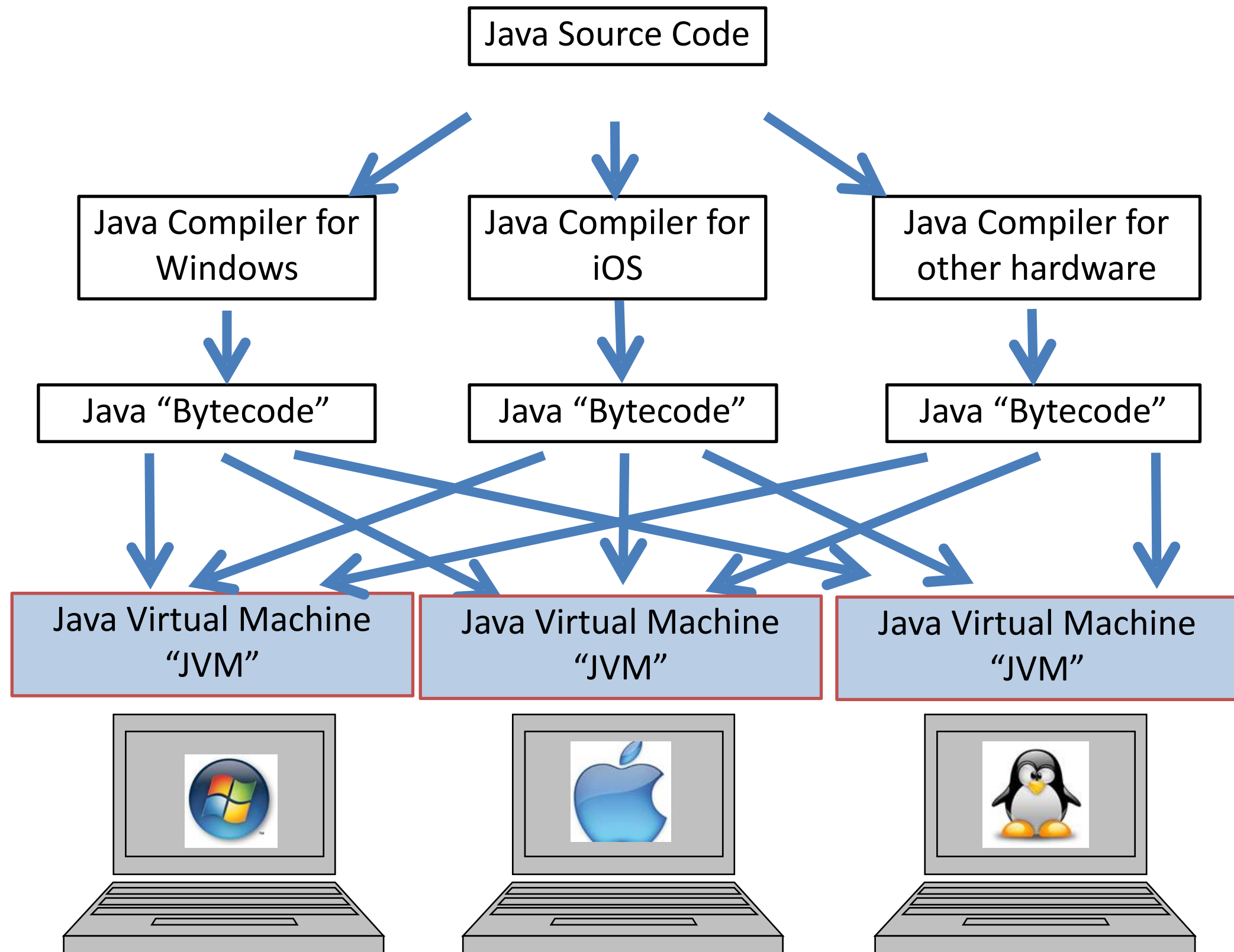
- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

Why Java?

Write Once; Run Anywhere?



Write Once; Run Anywhere!



The JDK and the JRE

API – Application Program Interface

Class libraries (MATH, GUI, Database...)

JRE – Java Runtime Environment

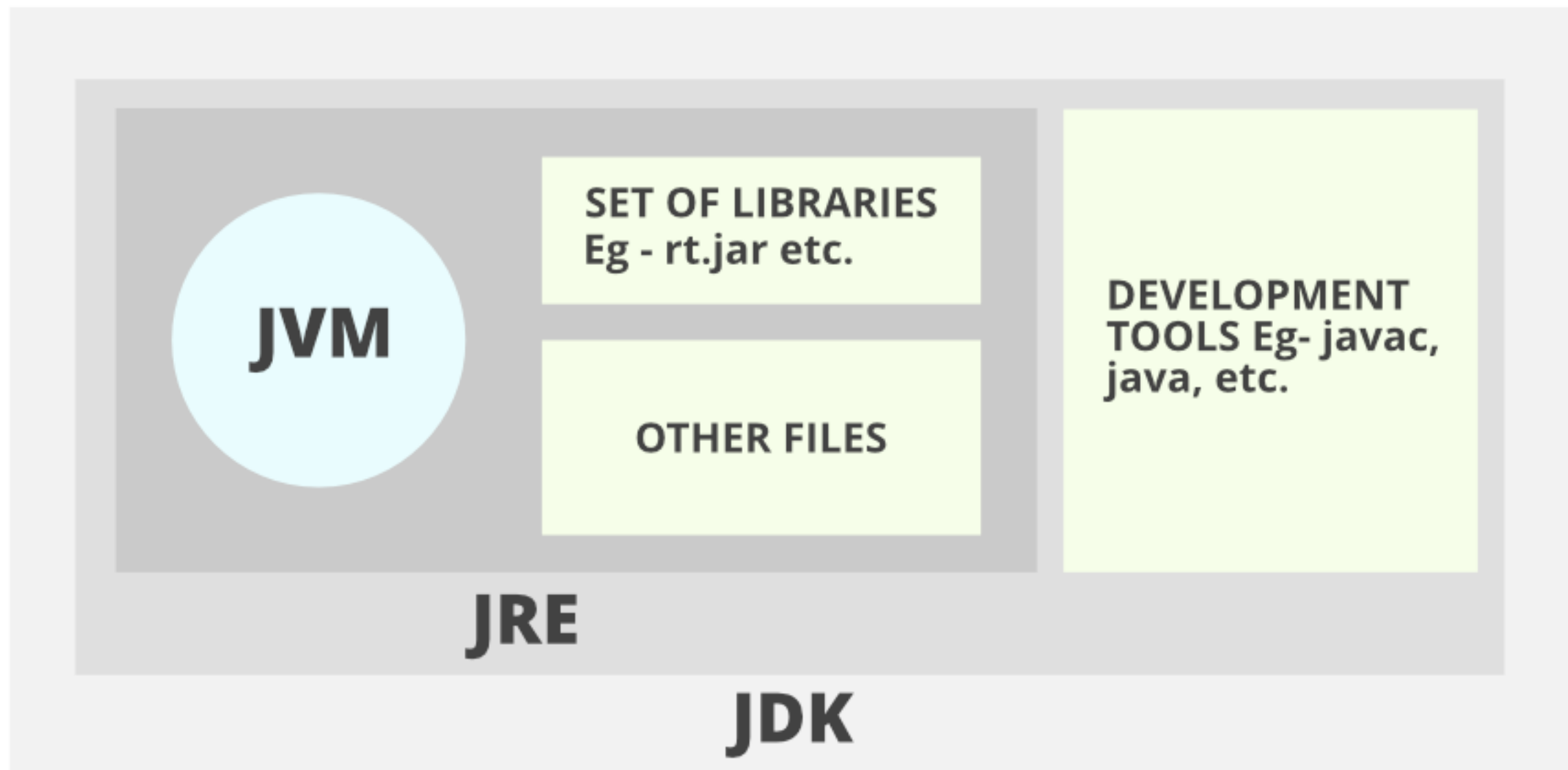
JVM plus API (can *run* Java programs)

JDK- Java Development Kit

JRE plus compiler, javadoc, ...

For ET 581 you need the JDK

The JDK and the JRE



Basic terminologies in Java

Java program is an object-oriented programming language, that means java is the collection of objects, and these objects communicate through method calls to each other to work together.

1. Class: The class is a blueprint (plan) of the instance of a class (object). It can be defined as a logical template that share common properties and methods.

2. Object: The object is an instance of a class. It is an entity that has behavior and state.

3. Method: The behavior of an object is the method.

4. Instance variables: Every object has its own unique set of instance variables. The state of an object is generally created by the values that are assigned to these instance variables.

Java Quickstart

In Java, every application begins with a class name, and that class must match the filename.

Let's create our first Java file, called Ex1.java, which can be done in any text editor (like Eclipse).

The file should contain a "Hello World" message, which is written with the following code:

The first Java program

```
public class Ex1 {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

Every line of code that runs in Java must be inside a class. In our example, we named the class Ex1. A class should always start with an uppercase first letter.

Note: Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

The main Method

- The main() method is required and you will see it in every Java program:
- **public static void main(String[] args)**
- Any code inside the main() method will be executed. You don't have to understand the keywords before and after main.
- For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

Java Source Code

Ex1.java



Java Compiler for
Windows

```
>javac Ex1.java
```

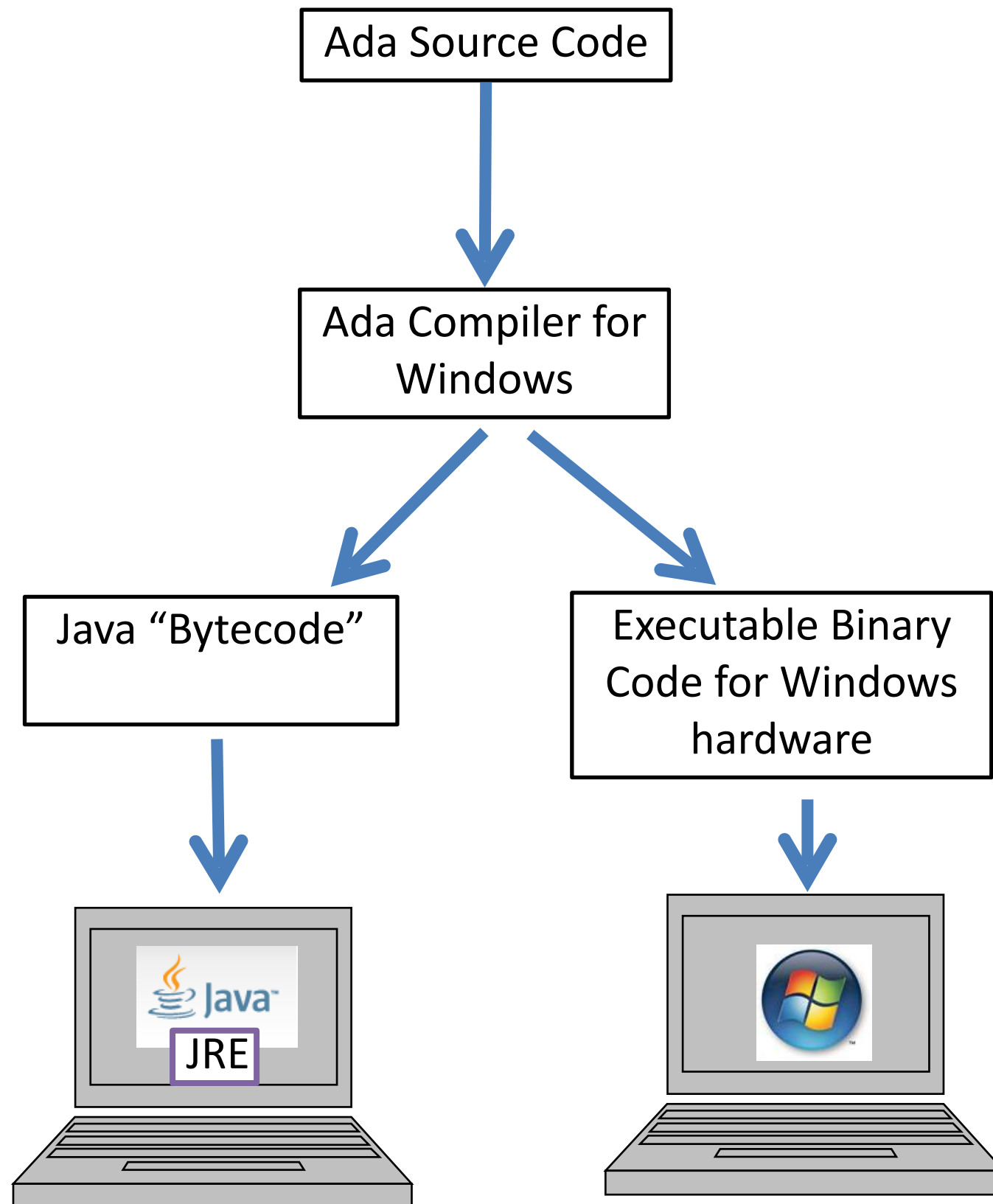


Java "Bytecode"

Ex1.class



```
>java Ex1
```



Prints

System.out.println()

- Inside the main() method, we can use the println() method to print a line of text to the screen:
- Note: The curly braces {} marks the beginning and the end of a block of code.
- Note: Each code statement must end with a semicolon.

Exercise

1. Write a script to Print the message “I am new to Java program!”.
2. Write a program that prints your own name, “Hello, xxx”.

Escape Characters

- Escape characters are used to manipulate text strings. They must always be inserted within the string itself (between double quotations).

\n — `System.out.print ("Hello\nWorld\n")`

\t — `System.out.print ("Hello\tWorld\n")`

Exercise

3. Print following text to the console, please use `'\t'` and `'\n'`.

Item	price
Apple	1.75
Orange	3.50
Banana	2.25
Total	9.00

4. Print following text to the console.

Name	ID
Tom Brown	15267789
My name is very long	12345678
David	99999999

Java Comments

- Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.
- Single-line Comments (`//`)
- Multi-line Comments (`/* ... */`)
- Documentation Comment (`/** ... */`)
- Normally, we use `//` for short comments, and `/* */` for longer.

Single-line Comments

- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by Java (will not be executed).
- Following example uses a single-line comment before a line of code:

```
// This is a comment  
System.out.println("Hello World");
```

Java Multi-line Comments

- Multi-line comments start with `/*` and ends with `*/`.
- Any text between `/*` and `*/` will be ignored by Java.
- Fowling example uses a multi-line comment (a comment block) to explain the code:

```
/* The code below will  
print the words Hello World  
to the screen, and it is amazing */
```

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

Variable

Variables

One of the most powerful features of a programming language is the ability to manipulate **variables**. A variable is a name that refers to a value.

A named location in memory into which you can place data. The program can read, write or alter these values as needed.

- In Java, you must declare a variable before using it.
- Allows storage of data internally for the program
- Allows storage of information from the user
- There are different types of variables services different needs

Declaring (Creating) Variables

- To create a variable, you must specify the type and assign it a value:
- Syntax: **type variable = value;**
- Where type is one of Java's types (such as int or String), and variable is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.

Name of Variables

- Rules for legal variable names
 1. Must begin with a letter, a currency symbol(\$) or an underscore character(_)
 2. Must not have any space in the name
 3. Can not have special characters
 - Only allow symbols are _ (underscore), and \$ (currency symbol)
 4. Must not be a keyword
 5. Variable names are case-sensitive
 6. Suggestions for variable names
 - Variable names should be meaningful
 - Variable names should be easy to read and write

Java Keyword

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

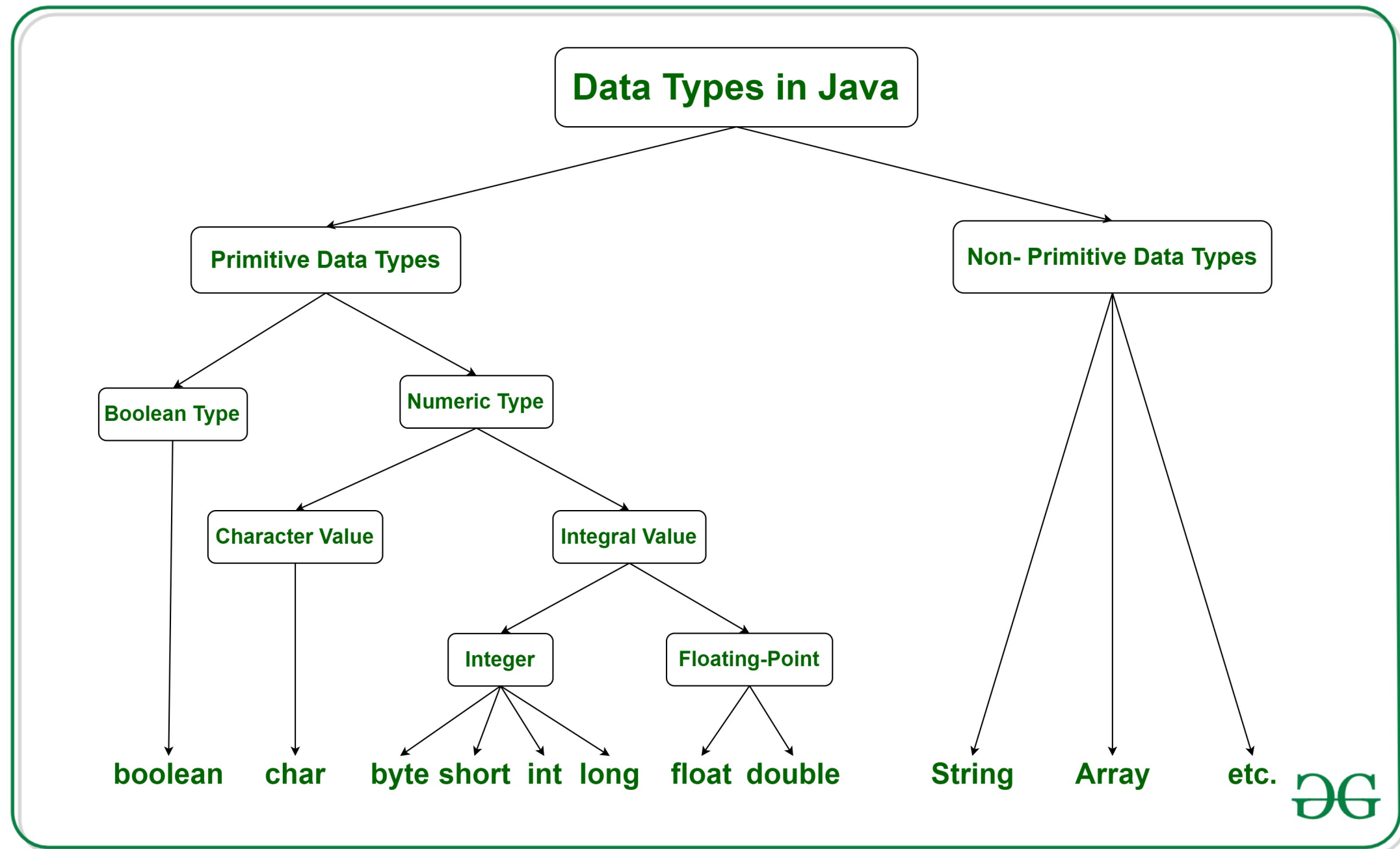
Assignment

- Now that we have declared variables, we want to use them to store values. We do that with an **assignment** statement.
- When you declare a variable, you create a named storage location.
- When you make an assignment to a variable, you update its value.
- As a general rule, a variable has to have the same type as the value you assign to it.
- Variables must be **initialized** (assigned for the first time) before they can be used. You can declare a variable and then assign a value later. You can also declare and initialize on the same line

Java Data Type

- **Data types in Java** are of different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. Java has two categories in which data types are segregated.
- **Primitive Data Type:** such as boolean, char, int, short, byte, long, float, and double
- **Non-Primitive Data Type** or **Object Data type:** such as String, Array, etc.

Java Data Type



Java Data Type

- Variables are containers for storing data values.
- In Java, there are different **types** of variables
- **int** - stores integers (whole numbers), without decimals, such as 123 or -123
- **float** - stores floating point numbers, with decimals, such as 19.99 or -19.99
- **char** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- **boolean** - stores values with two states: true or false
- **String** - stores text, such as "Hello". String values are surrounded by double quotes

Numbers

- Primitive number types are divided into two groups:
- **Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are **byte, short, int and long**. Which type you should use, depends on the numeric value.
- **Floating point types** represents numbers with a fractional part, containing one or more decimals. There are two types: **float and double**.
- Even though there are many numeric types in Java, the most used for numbers are int (for whole numbers) and double (for floating point numbers). However, we will describe them all as you continue to read.

int (integers)

- integer types
 - **byte (1 byte):** -128 to 127
 - **short (2 byte):** -32,768 to 32,767
 - **int (4 byte):** -2,147,483,648 to 2,147,483,647
 - **long (8 byte):** -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- Variable declaration
 - int day;
 - int month;
 - int year;
- Examples of integers
 - day= 1;
 - month= 2;
 - year= 2015;
- Examples of INVALID int
 - day= "1";
 - month="two";
 - year ='2014';
 - day= 1.5;

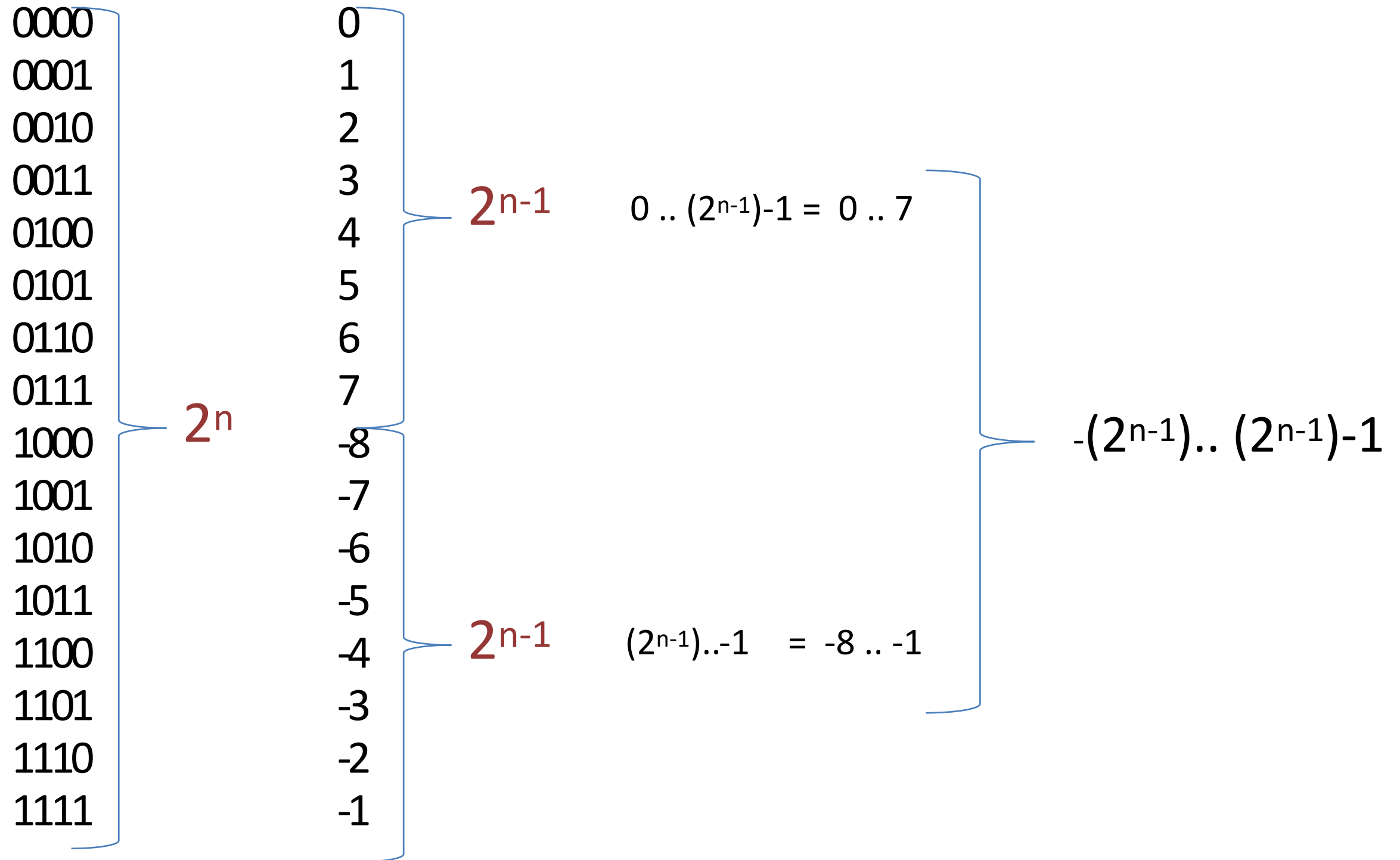
Primitives

Single-valued data items. Not Objects

byte	8-bit signed two's complement integer
short	16-bit signed two's complement integer
int	32-bit signed two's complement integer
long	64-bit signed two's complement integer
float	32-bit single-precision IEEE 754
double	64-bit single-precision IEEE 754
boolean	true/ false
char	16-bit Unicode character

The number of bits and the range of values

Two's complement integers, n=4



The number of bits and the range of values Two's complement integers, n=32

$$-(2^{n-1}).. (2^{n-1})-1$$

$$-(2^{32-1}).. (2^{32-1})-1$$

$$-(2^{31}).. (2^{31})-1$$

$$-2,147,483,648 .. +2,147,483,647$$

-OR-

$$2^{10} = 1,024 \text{ or about } 1,000 \text{ (Kilo)}$$

$$2^{20} = 1,024^2 \text{ or about } 1,000,000 \text{ (Mega)}$$

$$2^{30} = 1,024^3 \text{ or about } 1,000,000,000 \text{ (Giga)}$$

$$2^{40} = 1,024^4 \text{ or about } 1,000,000,000,000 \text{ (Tera)}$$

$$2^{31} = 2^1 * 2^{30}$$

$$= 2 \text{ G (about 2 billion)}$$

So, an integer can represent +/- 2 billion

floating point

- floating point types
 - **float (4 byte)**: Sufficient for storing 6 to 7 decimal digits
 - **double (8 byte)**: Sufficient for storing 15 decimal digits
- Variable declaration
 - double pi;
 - double e;
- Examples of double(decimals)
 - pi= 3.1415926535;
 - e=2.71828;
- Examples of INVALID double
 - pi= "3.141";
 - pi= '3.141';

char (characters)

- char (2 byte)
 - any single ASCII character
 - see ASCII table
- Variable declaration
 - char a;
 - char line;
 - char tag;
- Examples of char(characters)
 - a= 'a';
 - line='\n';
 - code= 80; // max number is 127 (ASCII)
- Example of INVALID char
 - tag= 234;
 - line="\n";

Why is the Size of char 2 bytes in Java?

- In other languages like C/C++ use only ASCII characters, and to represent all ASCII characters 8 bits is enough.
- But java uses the **Unicode system not the ASCII code system** and to represent the Unicode system 8 bits is not enough to represent all characters so java uses 2 bytes for characters.
- **Unicode** defines a fully international character set that can represent most of the world's written languages.

















char

ASCII 8-bit code $2^8 = 256$ chars




Unicode 16-bit code $2^{16} = 2^6 * 2^{10} = 64,000$ chars

Ç'ështëë Unicode?, in Albanian
የኒኮድ ምንድን ነው? in Amharic
ما هي التشفرة الموحدة "يونيكود" ؟ in Arabic
Ի՞նչ է Յունիկոդը ? in Armenian
ইউনিকোড কী? in Bangla
የኒኮድ ውረድ ግን? in Blin
Kakbo e Unicode ? in Bulgarian
什麼是Unicode(統一碼/標準萬國碼)? in T
什么是Unicode(统一码)? in Simplified Chi
Što je Unicode? in Croatian
Co je Unicode? in Czech
Hvad er Unicode? in Danish
Wat is Unicode? in Dutch
















<http://www.unicode.org/charts/>

	170	171
0	 1700	 1710
1	 1701	 1711
2	 1702	 1712
3	 1703	 1713
4	 1704	 1714
5	 1705	
6	 1706	
7	 1707	



Independent vowels

- 1700  TAGALOG LETTER A
 1701  TAGALOG LETTER I
 1702  TAGALOG LETTER U


Consonants

- 1703  TAGALOG LETTER KA
 1704  TAGALOG LETTER GA
 1705  TAGALOG LETTER NG
 1706  TAGALOG LETTER TA
 1707  TAGALOG LETTER DA
 1708  TAGALOG LETTER NA
 1709  TAGALOG LETTER PA
 170A  TAGALOG LETTER BA
 170B  TAGALOG LETTER MA
 170C  TAGALOG LETTER YA
 170D  <reserved>
 170E  TAGALOG LETTER LA
 170F  TAGALOG LETTER WA
 1710  TAGALOG LETTER SA
 1711  TAGALOG LETTER HA

Dependent vowel signs

- 1712  TAGALOG VOWEL SIGN I
 1713  TAGALOG VOWEL SIGN U

Virama

- 1714  TAGALOG SIGN VIRAMA

13A0

Cherokee

13FF

	13A	13B	13C	13D	13E	13F
0	D 13A0	F 13B0	G 13C0	† 13D0	‡ 13E0	β 13F0
1	R 13A1	Γ 13B1	Λ 13C1	∞ 13D1	∅ 13E1	ℳ 13F1
2	T 13A2	ℚ 13B2	h 13C2	R 13D2	P 13E2	ℏ 13F2
3	Ϣ 13A3	W 13B3	Z 13C3	ℓ 13D3	G 13E3	G ^w 13F3
4	ℴ 13A4	δ 13B4	‡ 13C4	W 13D4	V 13E4	B 13F4
5	i 13A5	ℙ 13B5	ℴ 13C5	§ 13D5	ℏ 13E5	
6	§ 13A6	G 13B6	T 13C6	ℴ 13D6	K 13E6	
7	ℴ 13A7	M 13B7	ℴ 13C7	ℴ 13D7	ℴ 13E7	

0F00

Tibetan

0FFF

	0F0	0F1	0F2	0F3	0F4	0F5	0F6	0F7	0F8	0F9	0FA	0FB	0FC	0FD	0FE	0FF
0																
1																
2																
3																
4																
5																
6																
7																
8																

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(88	58	1011000	130	X					
41	29	101001	51)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

boolean: true or false

- boolean (1 byte)
 - 1 or 0 represents true or false
 - Any number aside from 0 will evaluate as true.
- Variable declaration
 - `bool answer;`
 - `bool reply;`
- Examples of boolean
 - `answer = true;`
 - `answer = false;`
 - `reply = 0;`
 - `reply = 1;`
- Examples of INVALID boolean
 - `answer = "false";`
 - `reply = '0';`

Data Type Memory Allocation

Type	Size	Range
byte	1 bytes	-128 ~ 127
short	2 bytes	-32678 (-2^{15}) ~ 32767($2^{15}-1$)
int	4 bytes	-2,147,483,648 (-2^{31}) ~ 2,147,483,647($2^{31}-1$)
long	8 bytes	(-2^{63}) ~ 2,147,483,647($2^{63}-1$)
float	4 bytes	upto 7 decimal digits
double	8 bytes	upto 16 decimal digits
char	2 byte	ASCII chars (0 to 255)
boolean	1 byte	true or false

String Object

- The String type is so much used and integrated in Java, that some call it "the special **ninth** type".
- A String in Java is actually a **non-primitive** data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings. **Don't worry if you don't understand the term "object" just yet.** We will learn more about strings and objects later.

String

- String
 - Includes a variety of string manipulation options
- Variable declaration
 - String company;
 - String country;
 - String price;
- Examples of strings
 - company="IBM";
 - country="US";
 - price="3000";
- Examples of INVALID strings
 - company='IBM';
 - country=US;
 - price=3000;

Strings vs. Characters

- A character is a single symbol that takes up one column. Characters can be letters, digits, punctuation, whitespace or even unique symbols like arrows or happy faces.
- A string is a group of characters of any type, such as a word, sentence or random gibberish.

Arithmetic

Calculations

- Often when we develop programs calculations are embedded as part of it.
- It could be something as simple as counting number or something complex like simulating financial crisis.
- We need to be able to perform calculations!

Arithmetic

- **Operators** are symbols that represent simple computations. For example, the addition operator is +, subtraction is -, multiplication is *, and division is /.
- hour * 60 + minute is an **expression**, which represents a single value to be computed. When the program runs, each variable is replaced by its current value, and then the operators are applied. The values operators work with are called **operands**.

Arithmetic

Operation	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
()	Parenthese
++	Increment
--	Decrement

Modulus

- Finding the remainder of $5 / 2$
 - $5 \% 2 =$
- Find the remainder of the sum of two numbers divided by 2
 - A) $\text{number1} + \text{number2} \% 2 =$
 - B) $2 \% \text{number1} + \text{number2} =$
 - C) $(\text{number1} + \text{number2}) \% 2 =$
- Is the answer A, B or C?

Integer Division

- Integer division in Java will truncate any decimal value, for example:
 - $5 / 2 = 2$ for an integer division
 - $4 / 2 = 2$ this means $5 / 2 = 4 / 2$
 - $10 / 3 = 3$
 - $11 / 2 = 5$
- The resulting type is an Integer.
- What makes this an integer division?
 - Both the dividend and divisor are integers (not decimal).

Double (decimal) Division

- Double division in Java will retain the appropriate decimal value, for example:
 - $5/2.0=2.5$
 - $4/2.0=2$
 - $10/3.0= 3.333333\dots$
 - $11/ 2.0=5.5$
- The resulting type is a **Double**.
- What makes this an decimal division?
 - Either the dividend or divisor must be a decimal.

All Other Operations

- Same rule apply to Addition, Subtraction and Multiplication.
- If both of the values are of type int, result will be int.
- If either one of the type is double, result will be double.

Rounding errors

- Most floating-point numbers are only *approximately* correct. Some numbers, like reasonably-sized integers, can be represented exactly. But repeating fractions, like $1/3$, and irrational numbers, like π , cannot. To represent these numbers, computers have to round off to the nearest floating-point number.
- The difference between the number we want and the floating-point number we get is called **rounding error**.
- For many applications, like computer graphics, encryption, statistical analysis, and multimedia rendering, floating-point arithmetic has benefits that outweigh the costs. But if you need *absolute* precision, use integers instead.

Data Types and Order of Operation

- Look at the example below:

- $5.0 + 5 / 2$

1. What is the resulting data type of the first operation? -int
2. What is the resulting data type of the second operation? -double

- $5 + 5 / 2.0$

1. What is the resulting data type of the first operation? -double
2. What is the resulting data type of the second operation? -double

Type Casting

- If we want to convert from a **int to a double for a more** precise result we can do so with type casting.
- Type casting is a temporary change from one type to another.
- To type cast from int to double we can do the following:
- `double value= (double) 5/ 2; //value=2.5`

Java Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:
- Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double
- Narrowing Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

Widening casting

- Widening casting is done automatically when passing a smaller size type to a larger size type:

```
public class Main {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt;  
        // Automatic casting: int to double  
  
        System.out.println(myInt);  
        // Outputs 9  
        System.out.println(myDouble);  
        // Outputs 9.0  
    }  
}
```

Narrowing Casting

- Narrowing casting must be done manually by placing the type in parentheses in front of the value:

```
public class Main {  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble;  
        // Manual casting: double to int  
  
        System.out.println(myDouble);  
        // Outputs 9.78  
        System.out.println(myInt);  
        // Outputs 9  
    }  
}
```

Exercise

- Create double, float, int, short, long, byte variable each. Assign any values to double, float, byte variable.
- Convert between double -> short, float -> int, byte -> long.
- Print the result of short, int, long

Java Input

The System class

- We have been using **System.out.println** for a while, but you might not have thought about what it means. **System** is a class that provides methods related to the “system” or environment where programs run. It also provides **System.out**, which is a special value that provides methods for displaying output, including **println**.
- A **package** is a collection of related classes; **java.io** contains classes for “I/O” which stands for input and output.

Scanner class

- Java provides different ways to get input from the user. However, we will learn to get input from user using the object of Scanner class.
- In order to use the object of **Scanner**, we need to import **java.util.Scanner** package.
- Then, we need to create an object of the **Scanner** class. We can use the object to take input from the user.

Get Integer Input From the User

```
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = input.nextInt();
        System.out.println("You entered " + number);

        // closing the scanner object
        input.close();
    }
}
```

Input

- In the above example, we have created an object named input of the **Scanner** class. We then call the **nextInt()** method of the Scanner class to get an integer input from the user.
- Similarly, we can use **nextLong()**, **nextFloat()**, **nextDouble()**, and **next()** methods to get long, float, double, and string input respectively from the user.
- Note: We have used the **close()** method to close the object. It is recommended to close the scanner object once the input is taken.

Input Example

```
import java.util.Scanner;
class Input {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Getting float input
        System.out.print("Enter float: ");
        float myFloat = input.nextFloat();
        System.out.println("Float entered = " + myFloat);

        // Getting double input
        System.out.print("Enter double: ");
        double myDouble = input.nextDouble();
        System.out.println("Double entered = " + myDouble);

        // Getting String input
        System.out.print("Enter text: ");
        String myString = input.next();
        System.out.println("Text entered = " + myString);
    }
}
```

Exercise 5

Print the square of a number.

Goal:

Get a number and print the square.

Instruction:

0. Declare a variable “n”. (Use double type)
1. Get user input of a number and store the value input in the variable “n”.
2. Calculate the square of “n”.
3. Show the square of “n”.
4. Add comments in the code using //.

Exercise 6

- Declare one integer and one double
- Let user enter the integer
- Divide integer by 10 and store in the double
- Print out the double to the screen

Exercise 7

- Declare three variable
- Let the user enter first and second variable
- calculate the first int / second int, and save the result to third variable
- print out the third variable to the screen

Exercise 8

- Please enter a 4 digits integer
- Print out the digit by digit to the screen
- Example:
- Enter a 4 digit integer: 1234
- first digit: 1
- second digit: 2
- ...
- forth digit: 4

Program structure

- To review, a package is a collection of classes, which define methods. Methods contain statements, some of which contain expressions. Expressions are made up of **tokens**, which are the basic elements of a program, including numbers, variable names, operators, keywords, and punctuation like parentheses, braces and semicolons.
- The standard edition of Java comes with *several thousand* classes you can **import**, which can be both exciting and intimidating.

Literals and constants

- A value that appears in a program, like 2.54 (or " in ="), is called a **literal**. In general, there's nothing wrong with literals. But when numbers like 2.54 appear in an expression with no explanation, they make code hard to read. And if the same value appears many times, and might have to change in the future, it makes code hard to maintain.
- Values like that are sometimes called **magic numbers** (with the implication that being "magic" is not a good thing). A good practice is to assign magic numbers to variables with meaningful names

Formatting output

- System.out provides another method, called printf, that gives you more control of the format. The “f” in printf stands for “formatted”.
- The first value in the parentheses is a **format string** that specifies how the output should be displayed. This format string contains ordinary text followed by a **format specifier**, which is a special sequence that starts with a percent sign. The format specifier \%.3f indicates that the following value should be displayed as floating-point, rounded to three decimal places.
- The format specifier \%d displays integer values (“d” stands for “decimal”). The values are matched up with the format specifiers in order, so inch is displayed using \%d, and cm is displayed using \%f.

Exercise 9

Write a program that converts a temperature from Celsius to Fahrenheit. It should

- (1) prompt the user for input,
- (2) read a **double** value from the keyboard,
- (3) calculate the result, and
- (4) format the output to one decimal place.

For example, it should display "24.0 C = 75.2 F".

Here is the formula. Be careful not to use integer division!

$$F = C * 9/5 + 32$$

Exercise 10

- Write a program that converts a total number of seconds to hours, minutes, and seconds. It should
 - (1) prompt the user for input,
 - (2) read an integer from the keyboard,
 - (3) calculate the result, and
 - (4) use **printf** to display the output.
- For example, "5000 seconds = 1 hours, 23 minutes, and 20 seconds".
- *Hint:* Use the modulus operator.