Sept Batch 2024 Notes

Important Download Links

JDK Download URL:	https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html (Please sign up with oracle first and then download the JDK)
Eclipse Download URL:	https://www.eclipse.org/downloads/packages/
Setup java in Windows OS Environment variable :	https://mkyong.com/java/how-to-set-java_home-on-windows-10/

What is Java?

Topic Name	Details
Java &JR Eclipse Installation **(JDK/JRE/JVM Basics)**	Eclipse download link: https://www.eclipse.org/downloads/ (Any version of Eclipse will be fine) Or https://www.eclipse.org/downloads/packages/ Eclipse is an IDE(Integrated Development Environment) to develop applications with the help of Java, Python, C/C++, Ruby etc. Note: You need to download JDk first then after eclipse because eclipse is going to check if jdk installed or not. - You can check which jdk version is installed in your system by running the below command in terminal. (for mac/windows users) ---> java -version compiler converts entire code to byte code which is runnable in any machines (Windows, Linux, Mac, Unix) ☐ JVM - Java Virtual Machine (Converts byte code to machine code) JRE - Java Run Time Environment (Just runs java programs) Java is not fully object oriented because it supports primitive data types like **ch long, int, double** etc., which are not objects. Because in JAVA we use data types like int, float, double etc which are not object oriented, ar course is what opposite of OOP is. That is why **JAVA is not 100% object oriented**. Java - Platform Independent) JDK - Java Development Kit (JRE, JVM, Debugging, java docs)
	JDK - Java Development Kit comprises of = Tools + Compiler + Libs + JRE(Run Time env) + JVM(Platform Dependent) In a system, **JDK** helps in the **design and execution** of code. In contrast, **JRE** in a system simply helps in the **execution** of code, not in the d it. And JVM is always platform dependent.
Java Docs Link	Java Docs is the official Link to view the Interfaces , Classes etc in a given Java package Ex: Java.util - package If we want to know all the classes, interfaces of this package then java docs is helpful

How Java Source Code will be converted to .class file.

JDK (Java Development Kit) is a software development kit that contains the **JRE (Java Runtime Environment)**, compiler, and tools for deve Java applications. JRE (Java Runtime Environment) is a set of tools and libraries that allow Java applications to run on a computer. JVM (Jav Machine) is a virtual machine that executes Java code and provides runtime environment for Java applications.

In summary:

- JVM: runtime environment for executing Java code.
- JRE: includes JVM and libraries to run Java applications.
- JDK: includes JRE, compiler, and development tools for Java applications.

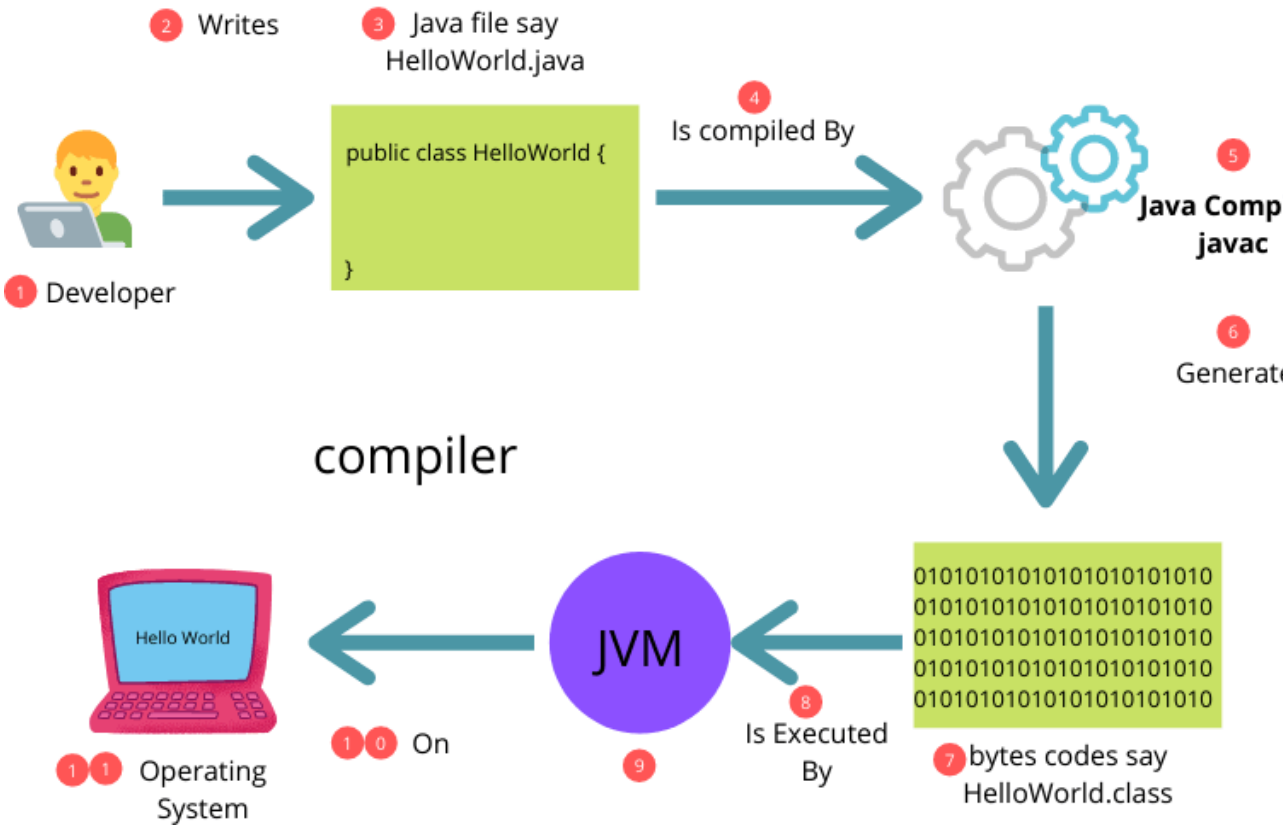
- [ ] The Java source code file (.java) is first compiled into Java bytecode (.class) using the Java compiler (javac). The bytecode is a platform-independent code that can be executed on any system with a Java Virtual Machine (JVM).

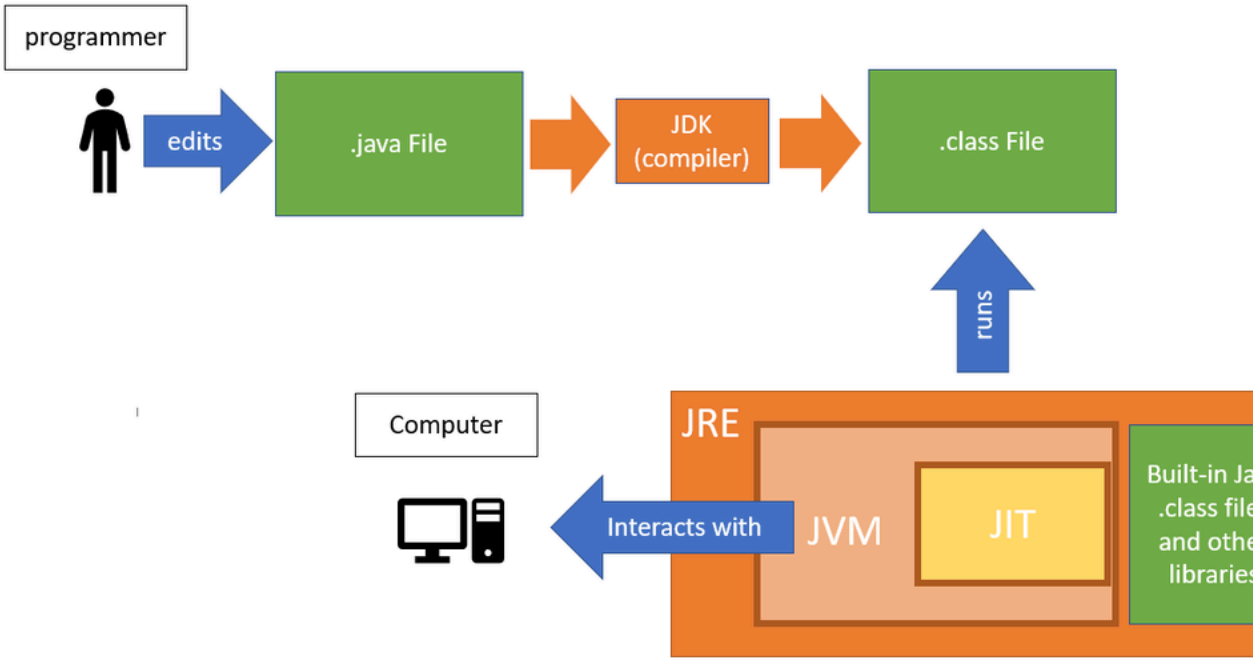
- When a Java program is executed, the JVM loads the bytecode into memory and runs it. The JVM interprets the bytecode and execute corresponding machine code instructions on the underlying system. The JVM also provides a runtime environment for the program, wh includes memory management, security, and other services.

- The JRE (Java Runtime Environment) contains the JVM and the Java class libraries, which provide a set of basic services and tools requ run a Java program. The JRE also includes the Java Plug-in, which enables Java applets to run in web browsers, and the Java Web Start, enables Java applications to be launched directly from the web.

In summary:

- Java compiler converts .java to .class bytecode.
- JVM runs .class bytecode using JRE.
- JRE provides runtime environment and class libraries for Java programs.
- JVM is platform dependent.



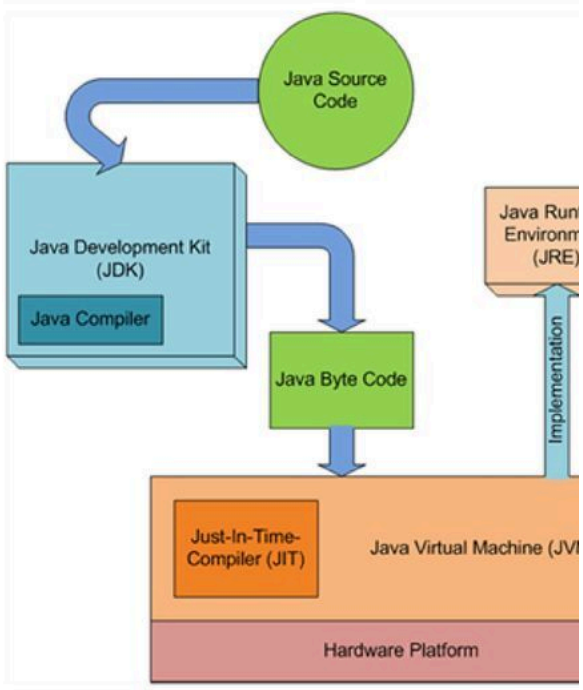


Difference between JDK/JRE/JVM/JIT

JVM Internals

Like a real computing machine, JVM has an instruction set and manipulates various memory areas at run time. Thus for different hardware platforms one has corresponding implementation of JVM available as vendor supplied JREs. It is common to implement a programming language using a virtual machine.

A Java virtual machine instruction consists of an opcode specifying the operation to be performed, followed by zero or more operands embodying values to be operated upon. From the point of view of a compiler, the Java Virtual Machine (JVM)is just another processor with an instruction set, Java bytecode, for which code can be generated. Life cycle is as follows, source code to byte code to be interpreted by the JRE and gets converted to the platform specific executable ones.



Jdk/jre/jit/jvm– Unit-I

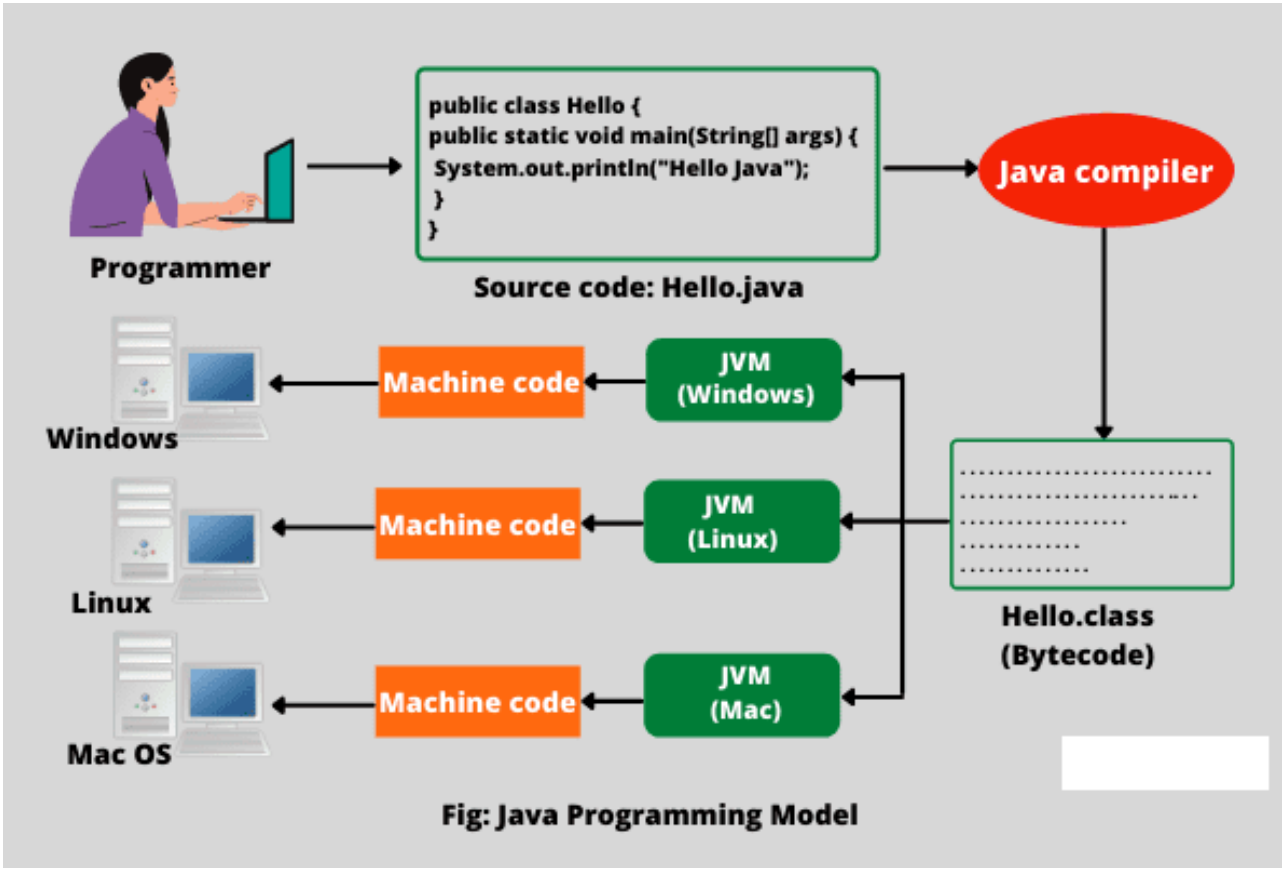
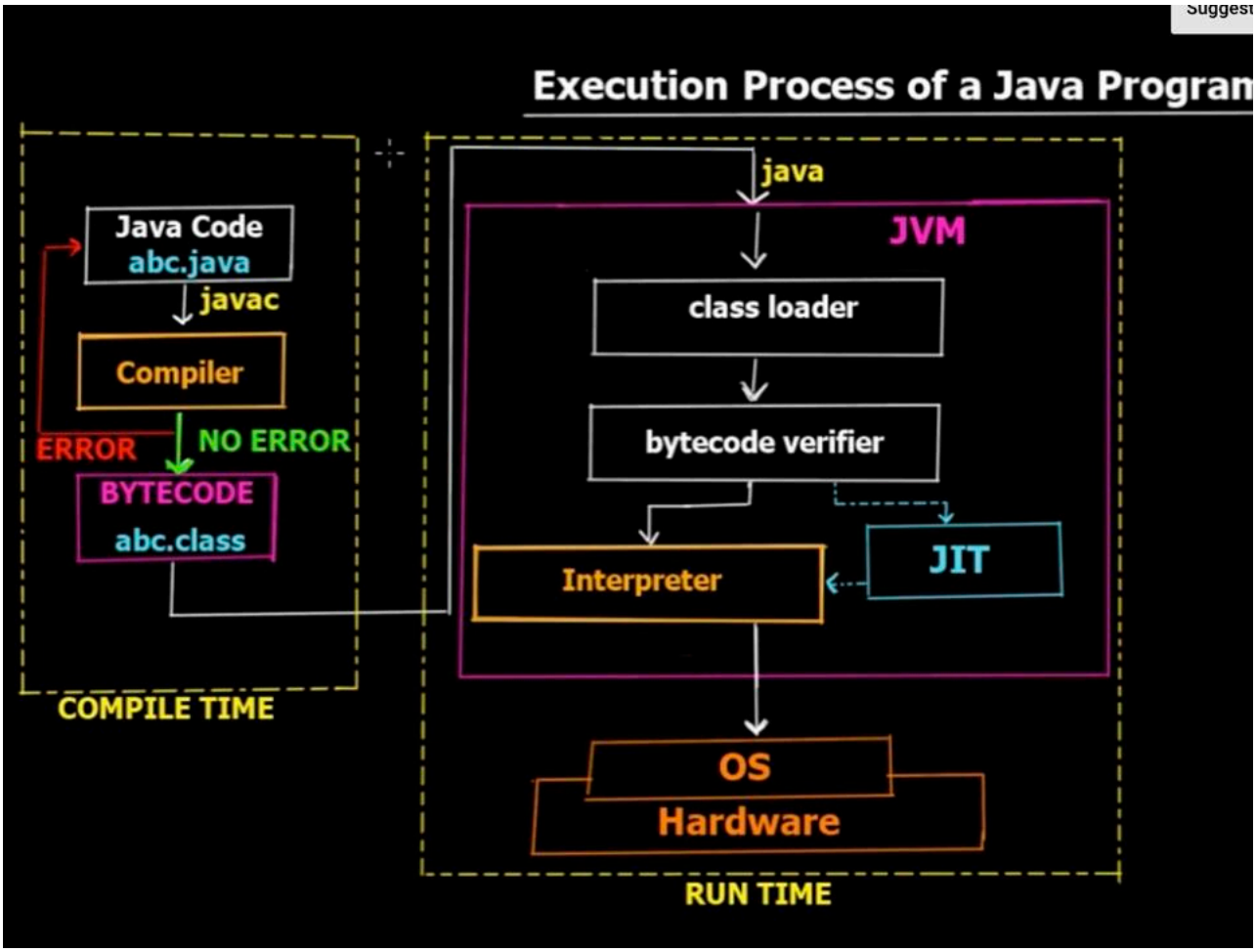
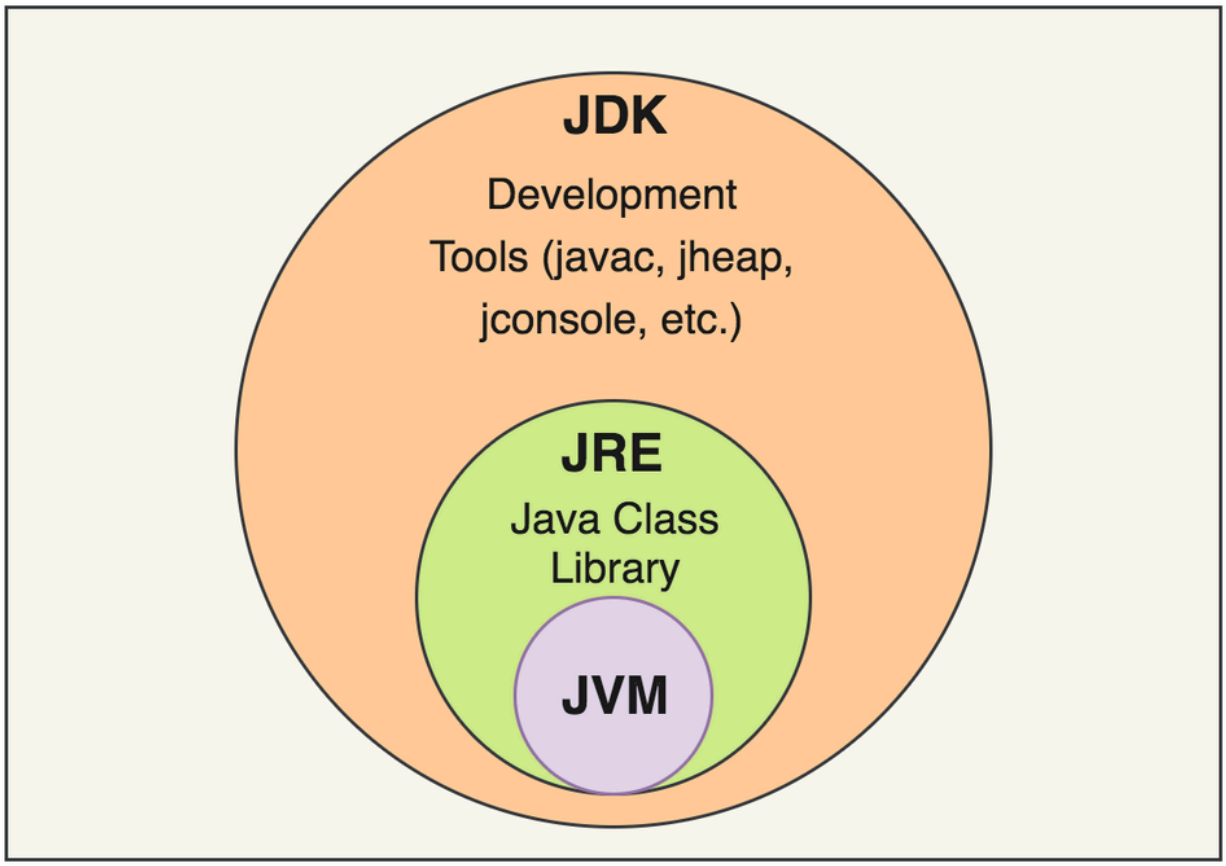


Fig: Java Programming Model



What is JIT (Just In Time)?	Just-In-Time (JIT) compilation is a part of the Java Runtime Environment (JRE) that improves the performance of Java applications by comp bytecodes into native machine code at runtime. Here's a breakdown of how it works and why it's beneficial: 1. **Bytecode**: Java programs are compiled into bytecode, which is a platform-independent code that can be executed by any Java Virtual M (JVM), regardless of the underlying hardware and operating system. 2. **JVM and Interpretation**: When a Java program is run, the JVM interprets the bytecode, converting it into machine code line by line as i executed. This process is straightforward but not very efficient in terms of execution speed. 3. **JIT Compilation**: To improve performance, the JIT compiler kicks in. It compiles the entire bytecode into native machine code that a co processor can execute directly. This compilation happens on the fly, just in time for execution. 4. **Performance Improvement**: The native machine code runs much faster than interpreted bytecode. JIT compilation thus improves the performance of Java programs after an initial warm-up period. 5. **HotSpot**: This is a term often associated with JIT compilation in Java. 6. The JIT compiler in the JVM focuses on the "hot spots" of the code—sections that are executed frequently—and compiles them into na code to maximize performance benefits.
Byte Code vs Machine Code	• Bytecode and machine code are two different forms of code that a computer can execute. • Bytecode is a platform-independent code that is generated by the Java compiler from Java source code. It is designed to be executed b Java Virtual Machine (JVM) and can run on any system with a JVM installed. Bytecode is an intermediate form of code that is not direc executable by the computer's hardware, but it is optimized for efficient execution by the JVM. • Machine code, on the other hand, is code that is directly executable by the computer's hardware. It is a low-level code that consists of instructions that correspond to specific operations the computer can perform. Machine code is platform-specific, meaning it is written specific type of processor and operating system and cannot run on other systems without modification. In summary: • Bytecode is platform-independent, optimized for JVM, and executed by JVM. • Machine code is platform-specific, low-level, and executed directly by the computer's hardware.
	When a program is compiled in languages like C or C++, it's translated directly into machine code specific to the target hardware and opera system. This makes these languages less portable than Java, as code compiled for one platform might not work on another without recompilation.
What is "WORA"	***Write once, run anywhere*** (***WORA***), or sometimes Write once, run everywhere (WORE) "Write once, run anywhere" (WORA) is a concept often associated with the Java programming language. It refers to the ability of Java to al developers to write code on one platform and have it run on any platform that has a compatible Java Virtual Machine (JVM) implementatic This is made possible by compiling Java source code into bytecode, which is a platform-independent intermediate representation of the co The bytecode is then executed by the JVM, which translates it into machine code specific to the underlying hardware and operating syster

Differences between JDK, JRE, and JVM:



Aspect	JDK (Java Development Kit)	JRE (Java Runtime Environment)	JVM (Java Virtual Machine)
Purpose	Development of Java applications, includes compiler and tools	Running Java applications	Execution of Java bytecode
Components	Compiler, debugger, libraries, development tools	Libraries and tools required to run Java applications	Runtime engine to execute Java bytecode
Includes JRE	Yes	Yes	No
Includes Compiler	Yes	No	No
Development Tools	Yes	No	No
Execution Environment	Yes	Yes	Yes
Platform Independence	Yes	Yes	No (JVM is platform-specific)
Example Usage	Writing, compiling, and testing Java applications	Running standalone Java applications	Interpreting and/or compiling bytecode

Topic Name	Details

JDK Path Setup in Windows	**To set up the JDK path in a Windows machine, follow these steps:** 1. Locate the JDK installation directory on your system. It is usually in the form of "C:\Program Files\Java\jdk-version". 2. Right-click on "This PC" or "My Computer" and select "Properties". 3. In the System Properties window, click on the "Advanced" tab and then click on the "Environment Variables" button. 4. Create a new variable : JAVA_HOME and its value: C:\Program Files\Java\jdk-version 5. In the Environment Variables window, under "System Variables", scroll down to find the "Path" variable and click on it to edit. 6. Click on the "Edit" and add this value: %JAVA_HOME%\bin 7. Click "OK" to close all windows and save the changes. 8. Open a new Command Prompt window and run the command "java -version" to confirm that the JDK path has been set up correctly. This will set up the JDK path for the current user.
	You can follow this article to setup the JDK path : https://mkyong.com/java/how-to-set-java_home-on-windows-10/

Topic Name	Details
create a Java project in Eclipse	**To create a Java project in Eclipse:** 1. Open Eclipse. 2. Click on File > New > Java Project. 3. Enter a project name and click Finish. 4. Right-click on the src folder in the Package Explorer and create a new package 5. Right-click on the package, go to New > Class. 6. Enter a class name and select public static void main(String[] args) in the Methods section. 7. Click Finish. 8. Write your Java code in the main method. 9. Save and run the project by clicking Run > Run As > Java Application.
create a Java project in IntelliJ IDEA **Download InteiiJ community version:** https://www.jetbrains.com/idea/download	**To create a Java project in IntelliJ IDEA:** 1. Open IntelliJ IDEA. 2. Click on Create New Project. 3. Select Java from the list of project types. 4. Choose a project SDK or click New to add a JDK. 5. Enter a project name and click Finish. 6. Right-click on the src directory and select New > Java Class. 7. Enter a class name and select public static void main(String[] args) in the Methods section. 8. Click OK. 9. Write your Java code in the main method. 10. Save and run the project by clicking Run > Run.

JavaSession: Topic : Data_Types

Data Types : There are 8 primitive data types in Java.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

--

Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false

Startertutorials.com

Data Types:

There are two data types:

1. Primitive Data Type: These datatypes don't need any object creation and they occupy the pre-defined memory (Memory Size is fixed)
2. Non Primitive Data Type: Where memory size is not fixed and object creation is needed

Ex: String, Array, Interface, Class

Difference between primitive data types and non primitive data types

Topic Name	Details

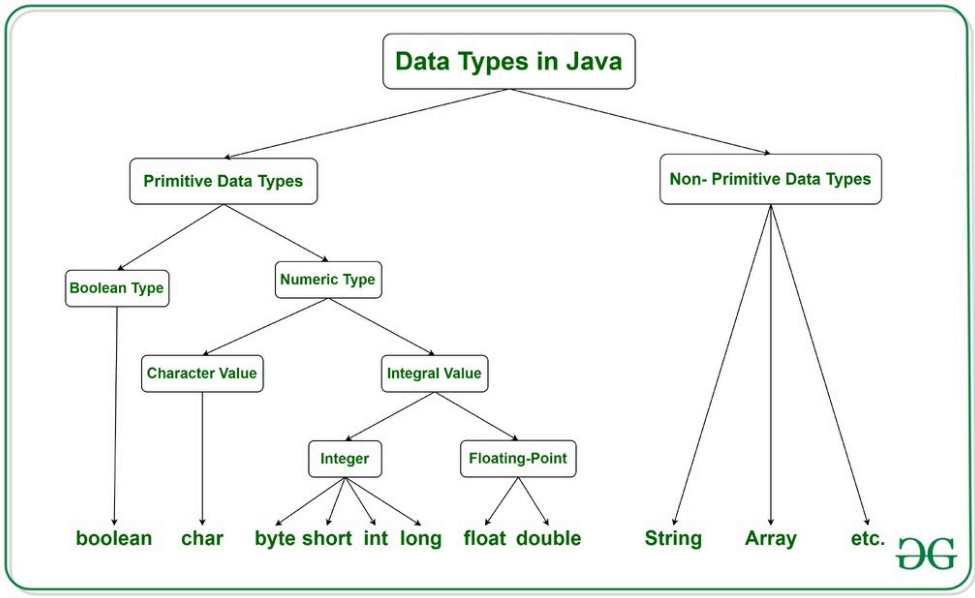
Difference between primitive data types and non primitive data types

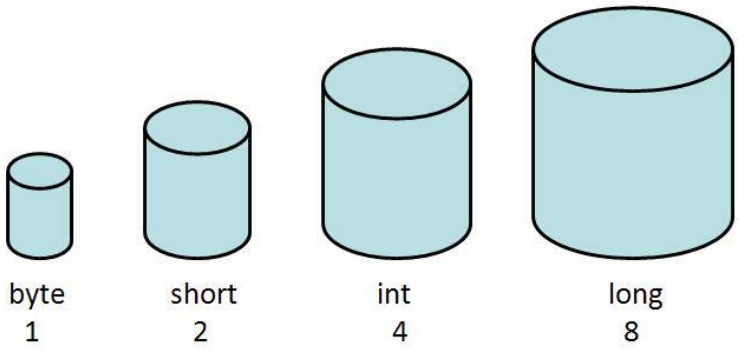
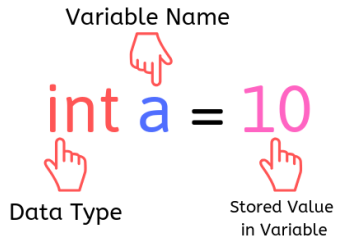
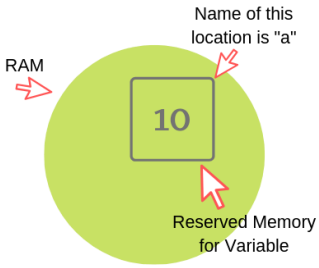
- Primitive data types in Java are the basic data types that are built-in to the language and represent simple values. Examples of primitive data types are int, float, double, char, and boolean. They are stored directly in memory, have a fixed size and a range of values, and can only be assigned values directly.
- Non-primitive data types, also known as reference data types, are data types that are constructed from primitive data types. Examples of non-primitive data types are arrays, classes, and strings. They are stored as references in memory and have a variable size, meaning they can hold an unlimited amount of data.
- They cannot be assigned values directly, but instead hold a reference to the object in memory where the value is stored.
- In summary, the main difference between primitive and non-primitive data types is that primitive data types represent simple values directly, while non-primitive data types represent objects that are stored in memory and hold references to the data.

Difference between primitive data types and non primitive data types

Properties	Primitive data types	Objects
Origin	Pre-defined data types	User-defined data types
Stored structure	Stored in a stack	Reference variable is stored in stack and the original object is stored in heap
When copied	Two different variables is created along with different assignment(only values are same)	Two reference variable is created but both are pointing to the same object on the heap
When changes are made in the copied variable	Change does not reflect in the original ones.	Changes reflected in the original ones.
Default value	Primitive datatypes do not have null as default value	The default value for the reference variable is null
Example	byte, short, int, long, float, double, char, boolean	array, string class, interface etc.

Java Data Types:



• 1 byte = 8 bits **How Memory allocated for a primitive data type:** **ex: int a = 10;**	  

Topic Name	Details
Why do we need to use L/l as a suffix for long numbers??	long myLong = 123456789L; The L suffix is required to write when the values to be stored is out of range of integer to tell compiler its long value because by default treated as int. We can write L for small values also but its not mandatory. Why do we need to use L/l as a suffix for long numbers?? Ans: This notation is used to make it explicit to the compiler and to other developers that the value should be treated as a **long** and not as an **int**. This is important when the value being assigned is outside the range of an **int** data type. Using the **L** notation for **long** data types is considered a best practice to ensure that the code is clear and readable.
Why do we need to use f as a suffix for float numbers??	float myFloat = 1.23f; This notation is used to make it explicit to the compiler and to other developers that the value should be treated as a **float** and not as a **double**. This is important when memory usage and precision need to be optimized, and when a smaller range of values is acceptable. Using the **f** notation for **float** data types is considered a best practice to ensure that the code is clear and readable.

Java naming conventions

Topic Name	Details

Java naming conventions

Java naming conventions are a set of rules for naming identifiers (such as variables, methods, classes, etc.) in the Java programming language.
The conventions help to keep the code consistent and easy to read, understand, and maintain. Some common Java naming conventions are:

1. Class names start with an uppercase letter and use PascalCase (e.g. EmployeeRecord, PaymentProcessor).
2. Method names start with a lowercase letter and use CamelCase (e.g. getEmployeeData, processPayment).
3. Variable names start with a lowercase letter and use CamelCase (e.g. employeeName, paymentAmount).
4. Constants are written in uppercase letters with words separated by an underscore (e.g. MAX_SIZE, MIN_WAGE).
5. Package names are all lowercase (e.g. java.util, com.example.payroll).
6. Interfaces start with an uppercase "I" followed by CamelCase (e.g. IEmployee, IPayment).
7. Acronyms are treated as a single word in names (e.g. HttpServlet, JdbcTemplate).
8. Enum names use CamelCase (e.g. Month, PaymentMethod).
9. Annotation names end with "Annotation" (e.g. OverrideAnnotation, DeprecatedAnnotation).
10. Use meaningful and descriptive names for identifiers, avoiding abbreviations and short names (e.g. "empName" instead of "en").yy,
11. Avoid using reserved words as identifier names (e.g. "class", "int").
12. Do not use Hungarian Notation, where the first few letters of an identifier indicate its type (e.g. "strName", "intAge").
13. Avoid using single-letter names except for temporary variables used within a small scope.
14. Try to keep the length of names reasonable, a
15. voiding excessively long or short names.
16. Use camelCase for local variables and method parameters.
17. Avoid using abbreviations unless they are widely recognized and well-established (e.g. "URL").
18. Prefix instance variables with "m" or "m_" (e.g. m_employeeName).
19. Prefix static variables with "s" or "s_" (e.g. s_maxCount).
20. Prefix boolean variables with "is" or "has" (e.g. isEmployee, hasData).
21. Use "get" and "set" prefixes for getter and setter methods (e.g. getEmployeeName, setEmployeeName).
22. Use the "toString" method to return a human-readable representation of an object.
23. Use a consistent naming convention throughout the project to maintain consistency and readability.

ASCII Table

Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C	Dec	Oct	Hex	C
0	0	0	^@	32	40	20		64	100	40	@	96	140	60	`
1	1	1	^A	33	41	21	!	65	101	41	A	97	141	61	a
2	2	2	^B	34	42	22	"	66	102	42	B	98	142	62	b
3	3	3	^C	35	43	23	#	67	103	43	C	99	143	63	c
4	4	4	^D	36	44	24	$	68	104	44	D	100	144	64	d
5	5	5	^E	37	45	25	%	69	105	45	E	101	145	65	e
6	6	6	^F	38	46	26	&	70	106	46	F	102	146	66	f
7	7	7	^G	39	47	27	'	71	107	47	G	103	147	67	g
8	10	8	^H	40	50	28	(	72	110	48	H	104	150	68	h
9	11	9	^I	41	51	29	)	73	111	49	I	105	151	69	i
10	12	a	^J	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	b	^K	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	c	^L	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	d	^M	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	e	^N	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	f	^O	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	^P	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	^Q	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	^R	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	^S	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	^T	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	^U	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	^V	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	^W	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	^X	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	^Y	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	^Z	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	^[	59	73	3b	;	91	133	5b	[	123	173	7b	{
28	34	1c	^\	60	74	3c	<	92	134	5c	\	124	174	7c	|
29	35	1d	^]	61	75	3d	=	93	135	5d	]	125	175	7d	}
30	36	1e	^^	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	^_	63	77	3f	?	95	137	5f	_	127	177	7f	

ASCII Range for Letters and Numbers:	For lowercase letters (a-z), the ASCII range is 97 to 122. For uppercase letters(A-Z), the ASCII range is 65 to 90. For numerals(0-9), the ASCII range is 48 to 57.
Unicode and ASCII Characters	Unicode and ASCII are character encoding standards used to represent text characters as numerical values. They provide a way for computers to understand and exchange textual information by assigning a unique number (code point) to each character. **ASCII (American Standard Code for Information Interchange):** ASCII is one of the earliest character encoding standards and is primarily used for representing English characters and control characters in computers. It uses 7 bits to represent 128 different characters, including letters, numbers, punctuation, and control characters. Here are a few ASCII character examples along with their decimal and binary representations: • Character: 'A', Decimal: 65, Binary: 01000001 • Character: 'a', Decimal: 97, Binary: 01100001 • Character: '0', Decimal: 48, Binary: 00110000 • Character: '!', Decimal: 33, Binary: 00100001 **Unicode:** Unicode is a more comprehensive character encoding standard that aims to cover characters from all writing systems in the world, including various languages, symbols, and emojis. It uses a larger number of bits (usually 16 or 32 bits) to represent a much wider range of characters compared to ASCII. Here are a few Unicode character examples along with their hexadecimal and binary representations: • Character: 'A', Unicode Hex: U+0041, Binary: 01000001 (Equivalent to ASCII for ASCII characters) • Character: '€', Unicode Hex: U+20AC, Binary: 00100000 10101100 (Euro sign) • Character: 'न', Unicode Hex: U+0928, Binary: 10010010 10001000 (Devanagari letter 'na' in Hindi) • Character: '😊', Unicode Hex: U+1F60A, Binary: 00011111 01100000 00101000 00101010 (Smiling face emoji) Unicode provides a way to represent a vast array of characters from various languages and cultural contexts, making it more suitable for global communication and content.

All Divide By Zero cases

Cases	Results	Example	Tricky examples
Any integer divided by 0	Arithmetic Exception	9/0.	0/0 is also exception
0 divided by Any Integer.	0	0/9=0	
Any integer or Float divided by 0 or 0.0	Infinity	2.5/0,2.5/0.0	9/0.0 = infinity 9.5/0.0=infinity
int divided by int	integer	5/2=2	
Int divided by float or float divide by integer	float	5.0/2=2.5 , 5/2.0=2.5	
Int 0 divided by float 0 or float 0 divided by int 0	NaN- Not a number	0/0.0=naN 0.0/0=naN 0.0/0.0=naN	

String Concatenation	In Java, a **String** is a in-built class that represents a sequence of characters. It is one of the most widely used classes in Java and is used to represent text data. Ex: **String s = "Hello World";** **String msg = "This is my java code";** ***String Concatenation:*** String concatenation in Java refers to the operation of combining two or more strings into a single string. This can be done using the "+" operator. The "+" operator can be used to concatenate not just strings, but also other data types such as numbers, which will be automatica converted to strings before concatenation. **Examples of string concatenation using the "+" operator:** ***int*** **a** = 10, int **b** = 20; • *System.out.println* (a+b); - Output= **30** • Here '+' sign is behave as Arithmetic Operator • *System.out.println* (a+b+" test data"); Output= "**30test data**" • Here 1st '+' sign is behave as Arithmetic Operator, while 2nd '+' sign Concatenation Operator • *System.out.println* ("test data" + a); Output= "**test data10**" • Here '+' sign is behave as Concatenation Operator • *System.out.println* ("test data" + a+b) Output= "**test data1020**" • Here '+' sign is behave as Concatenation Operator • **Note** - Since execution happens left to right , the placement of the string matters while printing to get the correct data • **System** is a class ***out*** is variable and **println()** is method.

Incremental/Decremental Operators:

1) What will be the output of the following program?

```
public class IncrementDecrementQuiz
{
    public static void main(String[] args)
    {
        int i = 11;

        i = i++ + ++i;
```

2) Guess the output of the following program?

```
public class IncrementDecrementQuiz
{
    public static void main(String[] args)
    {
        int a=11, b=22, c;
```

```
6
7           c = a + b + a++ + b++ + ++a + ++b;
```

3) What will be the output of the below program?

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

      i = i++ - --i + ++i - i--;

      System.out.println(i);

   }
}
```

4) Is the below program written correctly?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
      {
         boolean b = true;

         b++;

         System.out.println(b);
   }
}
```

5) What will be the output of the below program?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int i=1, j=2, k=3;

      int m = i-- - j-- - k--;

      System.out.println("i="+i);
      System.out.println("j="+j);
      System.out.println("k="+k);
      System.out.println("m="+m);
   }
}
```

6) What will be the output of the following program?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int a=1, b=2;
      System.out.println(--b - ++a + ++b - --a);

   }
}
```

7) What will be the value of i, j and k in the below program?

```
public class IncrementDecrementQuiz
```

```
{
   public static void main(String[] args)
   {
•          int i=19, j=29, k=0;
•
•         int m = i-- - j-- - k--;
•
•         System.out.println("i="+i);
•         System.out.println("j="+j);
•         System.out.println("k="+k);
   }
}
```

8) What is wrong with the below program? Why it is showing compilation error?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int i = 11;

      int j = --(i++);
   }
}
```

9) Guess the value of *p* in the below program?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int m = 0, n = 0;

      int p = --m * --n * n-- * m--;

      System.out.println(p);
   }
}
```

10) What will be the output of the following program?

```
public class IncrementDecrementQuiz
{
•     public static void main(String[] args)
   {
      int a=1;

      a = a++ + ++a * --a - a--;

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

11) What will be the outcome of the below program?

```
public class IncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int a = 11++;

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

12) What will be the outcome of the below program?

```
public class JavaIncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      int ch = 'A';

      System.out.println(ch++);
   }
}
```

13) **What will be the outcome of the below program?**

```
public class JavaIncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      char ch = 'A';

      System.out.println(++ch);
   }
}
```

14) **What will be the outcome of the below program?**

```
public class JavaIncrementDecrementQuiz
{
   public static void main(String[] args)
   {
      double d = 1.5, D = 2.5;

      System.out.println(d++ + ++D);//1.5 + 3.5 = 5.0
   }
}
```

Type Casting:

Type Casting

Types of Data Types

Primitive or Fundamental Data Types
Referenced or Advanced Data Types

Casting Primitive Data Type

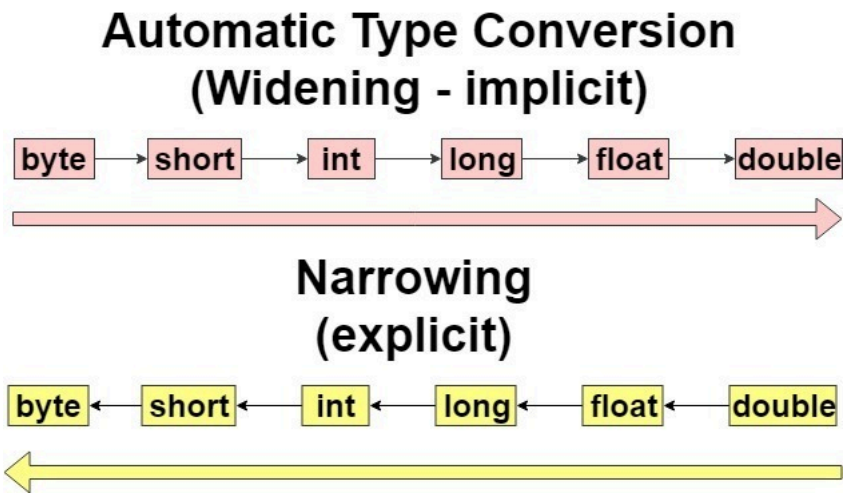
- **Widening:** Converting a lower data type into higher data type is called widening.
- **Narrowing:** Converting a higher data type into lower data type is called narrowing

Type Casting:		

Widening or Automatic Type Conversion

In Java, when assigning a value of a smaller data type to a larger data type, Java automatically converts the smaller data type into the larger one. This is known as automatic or widening type conversion and it doesn't require explicit casting. The data types are arranged from smallest to largest as follows:

1. **byte** - This is an 8-bit signed integer.
2. **short** - This is a 16-bit signed integer.
3. **int** - This is a 32-bit signed integer.
4. **long** - This is a 64-bit signed integer.
5. **float** - This is a 32-bit IEEE 754 floating point.
6. **double** - This is a 64-bit IEEE 754 floating point.



For example, here's how automatic conversion works:

```
byte byteValue = 42;
short shortValue = byteValue; // automatic conversion from byte to short
int intValue = shortValue; // automatic conversion from short to int
long longValue = intValue; // automatic conversion from int to long
float floatValue = longValue; // automatic conversion from long to float
double doubleValue = floatValue; // automatic conversion from float to double
```

Narrowing or Explicit Type Conversion

Conversely, when assigning a value of a larger data type to a smaller data type, Java requires explicit casting. This is because narrowing conversion could potentially lead to data loss, and Java needs you to confirm you understand this risk by writing a cast. The order for narrowing conversion is the reverse of widening conversion.

Here's an example of explicit casting:

```
double doubleValue = 42.0;
float floatValue = (float) doubleValue; // explicit cast from double to float
long longValue = (long) floatValue; // explicit cast from float to long
int intValue = (int) longValue; // explicit cast from long to int
short shortValue = (short) intValue; // explicit cast from int to short
byte byteValue = (byte) shortValue; // explicit cast from short to byte
```

Rules and Considerations

- Widening conversions do not lose information about the overall magnitude of a numeric value.
- Narrowing conversions may lose information about the overall magnitude of a numeric value and may also lose precision.
- When converting from floating-point to integral types, the fractional part is truncated.
- Casting in the opposite direction (narrowing conversion) requires explicit casts.
- Java does not allow casting boolean values to any numeric type and vice versa.

Remember that while widening conversions are generally safe, narrowing conversions should be done

		carefully to avoid data loss.

Widening Conversion Examples	// Example 1: From integer to floating-point types
	int myInt = 100;
	long myLong = myInt; // No casting required
	float myFloat = myLong; // No casting required
	double myDouble = myInt; // No casting required - int to double is also widening
	
	System.out.println("Int value: " + myInt);
	System.out.println("Long value: " + myLong);
	System.out.println("Float value: " + myFloat);
	System.out.println("Double value: " + myDouble);
	
	// Example 2: From char to integral types
	char myChar = 'A';
	int charToInt = myChar; // char to int is widening since char is 16-bit and int is 32-bit
	
	System.out.println("Char value: " + myChar);
	System.out.println("Int value from char: " + charToInt); // Outputs 65, which is ASCII value of 'A'	

Narrowing Conversion Examples	// Example 1: From floating-point to integral types
	double myDoubleValue = 9.78;
	int myIntValue = (int) myDoubleValue; // Explicit casting is required
	
	System.out.println("Double value: " + myDoubleValue);
	System.out.println("Converted Integer value: " + myIntValue); // Outputs 9, fractional part is lost
	
	// Example 2: From long to smaller types
	long myLongValue = 123456789L;
	int myIntFromLong = (int) myLongValue; // Explicit casting is required
	short myShortFromLong = (short) myLongValue; // Explicit casting is required
	byte myByteFromLong = (byte) myLongValue; // Explicit casting is required
	
	System.out.println("Long value: " + myLongValue);
	System.out.println("Converted Int value: " + myIntFromLong); // Potential loss of magnitude
	System.out.println("Converted Short value: " + myShortFromLong); // Potential loss of magnitude
	System.out.println("Converted Byte value: " + myByteFromLong); // Potential loss of magnitude
	
	// Example 3: From char to byte
	char myCharacter = 'F';
	byte myByteFromChar = (byte) myCharacter; // Explicit casting is required
	
	System.out.println("Character value: " + myCharacter);
	System.out.println("Converted Byte value: " + myByteFromChar); // Outputs 70, which is ASCII value of 'F'	