

Key Points

- James Gosling, Mike Sheridan, and Patrick Naughton commenced the Java language project in June 1991.
- The small team of sun engineers called Green Team.
- The first name of java is “Green Talk” later changed to “OAK” which was changed to “JAVA” in the year 1995.
- The first extension of java program is ‘.gt’
- The first browser which was developed using java language is “Hot Java Browser”.
- The first version of java is JDK 1.0 and the latest version is Java SE 21

There is a team of developers, they need to create software which will work along with hardware devices by using traditional programming languages like c and c++

Example: washing machines

Because of using **pointer** concept the software is **crashed** every time

What is a pointer

This is a concept which is presented in C and C++

Pointers holds address locations

So, to avoid the crash of software they need to create a new programming language named 'Green talk' after that its name has changed to OAK then it is changed to JAVA

The first extension of our java program is ".gt" then changed to ".java"

C Vs C++ Vs Java			
Feature	C	C++	Java
Developed By (&) Year	Dennis Ritchie 1972	Bjarne Stroustrup 1979	James Gosling 1991
Model	Procedural	Object Oriented	Object Oriented
Platform Dependency	Dependent	Dependent	Independent
Keywords	32	63	50 (goto & const are reserved keywords)
Pre-processor directives	Supported (#include, #define)	Supported (#include, #define)	Not Supported (import is supported)
Inheritance	Not Supported	Supported	Multiple Inheritance is not supported
Pointers	Supported	Supported	Eliminated & are replaced with references
Note: reference is the address of the memory location where the object is stored			

<u>Features</u>	C	C++	Java
Developed By	Dennis Ritchie	Bjarne Stroustrup	James Gosling
Model	POP	OOP	OOP
Platform Dependency	Dependent	Dependent	In-Dependent Write-Once-Run-Anywhere
Keywords	32	63	Above 50 <div> <div> true false null </div> ⇒ These are NOT Keywords, they are values (Literals) </div>
Pre-Processor Directives	<div> <div>#include #define</div> ⇒ Supports </div>	<div> <div>#include #define</div> ⇒ Supports </div>	<div> <div>#include #define</div> ⇒ Does NOT Supports In Java we are having 'import' statement. </div>
Inheritance	DOES NOT SUPPORTS	SUPPORTS	SUPPORTS Java does not support Multiple inheritance through classes , but we can achieve this multiple inheritance with the help of interfaces
Pointers	Supports	Supports	Pointers are eliminated from Java and those are replaced with references

POP (procedure Oriented Programing languages)

Maily concentrates on step-by-step flow of executions

Example first it executes

Method 1

Method 2

Method 3

Method 4

OOP (object-oriented programming language)

It mainly concentrates on data security

Here we can execute any method as per our requirements

Java 8 version features had a huge impact, it is a revolted version.

The new features in java 8 are

Lambda expression

Functional interfaces

Stream API

Method references

New date time API

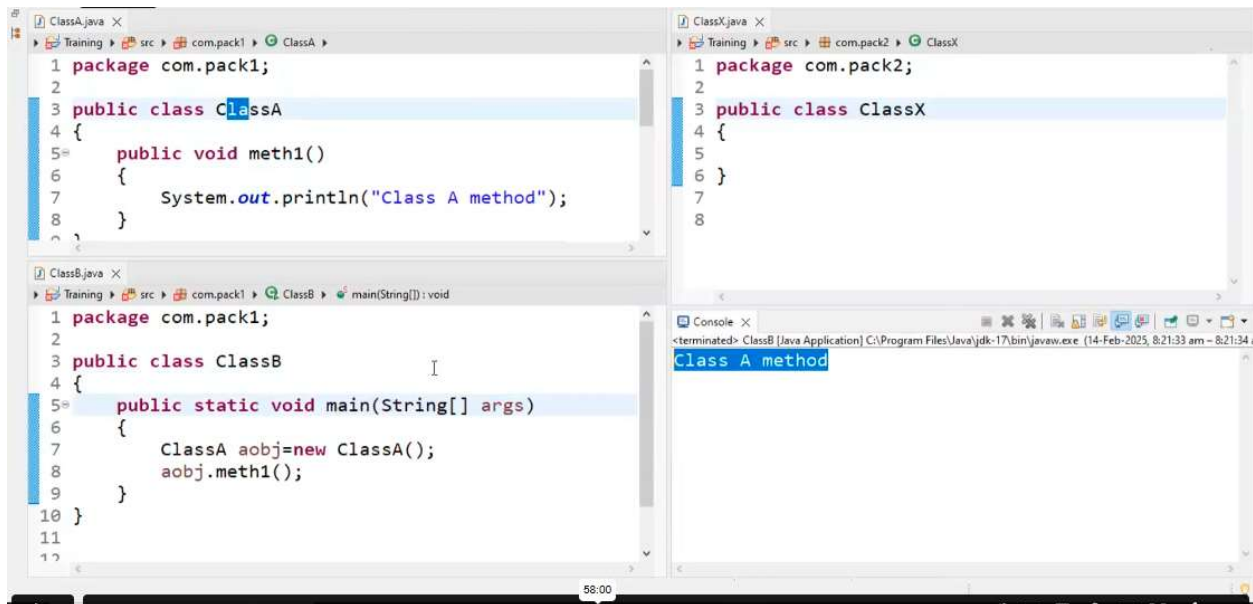
Parallel array sorting

Optional class ...etc.

Java version History

Version	Release Date		
JDK 1.0	January 23, 1996		
JDK 1.1	February 19, 1996		
J2SE 1.2	December 8, 1998		
J2SE 1.3	May 8, 2000		
J2SE 1.4	February 6, 2002		
J2SE 5.0 ✓	September 30, 2004		
Java SE 6	December 11, 2006		
Java SE 7	July 28, 2011		
Java SE 8 ✓	March 18, 2014		
Java SE 9	September 21, 2017		
Java SE 10	March 20, 2018		
Java SE 11 ✓	September , 2018		
Java SE 12	March 19, 2019	Java SE 15	September 2020
Java SE 13	September 17, 2019	Java SE 16	March 2021
Java SE 14	March 17, 2020	Java SE 17	September 2021

Below run program in ClassB



The screenshot shows an IDE with three windows. The top-left window shows `ClassA.java` with the following code:

```
1 package com.pack1;
2
3 public class ClassA
4 {
5     public void meth1()
6     {
7         System.out.println("Class A method");
8     }
9 }
```

The bottom-left window shows `ClassB.java` with the following code:

```
1 package com.pack1;
2
3 public class ClassB
4 {
5     public static void main(String[] args)
6     {
7         ClassA aobj=new ClassA();
8         aobj.meth1();
9     }
10 }
11
```

The top-right window shows `ClassX.java` with the following code:

```
1 package com.pack2;
2
3 public class ClassX
4 {
5
6 }
7
8
```

The bottom-right window is the console, showing the output: `Class A method`.

Note

If the both the classes (classA, classB) are in the same package we can use those classes according to our requirements. No problem with this.

But if both the classes are not in same package we should 100% need to use Import.

Below run the program in ClassX

```
1 package com.pack1;
2
3 public class ClassA
4 {
5     public void meth1()
6     {
7         System.out.println("Class A method");
8     }
9 }

1 package com.pack1;
2
3 public class ClassB
4 {
5     public static void main(String[] args)
6     {
7         ClassA aobj=new ClassA();
8         aobj.meth1();
9     }
10 }
11

1 package com.pack2;
2
3 import com.pack1.ClassA;
4
5 public class ClassX
6 {
7     public static void main(String[] args)
8     {
9         ClassA aobj=new ClassA();
10        aobj.meth1();
11    }
12 }
13
14
```

Console
<terminated> ClassX [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (14-Feb-2025, 8:30:05 am) [pid: 11]
Class A method

```
1 package com.pack1;
2
3 public class ClassA
4 {
5     public void meth1()
6     {
7         System.out.println("Class A method");
8     }
9 }

1 package com.pack1;
2
3 public class ClassB
4 {
5     public void meth2()
6     {
7         System.out.println("Class B method");
8     }
9 }
10
11 public static void main(String[] args)
12 {
13     ClassA aobj=new ClassA();
14 }

1 package com.pack2;
2
3 //import com.pack1.ClassA;
4 //import com.pack1.ClassB;
5
6 import com.pack1.*;
7
8 public class ClassX
9 {
10     public static void main(String[] args)
11     {
12         ClassA aobj=new ClassA();
13         aobj.meth1();
14
15         ClassB bobj=new ClassB();
16         bobj.meth2();
17     }
18 }

Console  
<terminated> ClassX [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (14-Feb-2025, 8:30:06 am) [pid: 11]  
Class A method
```

By using the above statement **import com.pack1*;**

In this case all the classes in com.pack1 will be imported

This is not recommended because we not required all classes instead of import only required classes.

```
1 package com.pack1;
2
3 public class ClassA
4 {
5     public void meth1()
6     {
7         System.out.println("Class A method");
8     }
9 }
10
```

```
1 package com.pack1;
2
3 public class ClassB
4 {
5     public void meth2()
6     {
7         System.out.println("Class B method");
8     }
9 }
10
11 public static void main(String[] args)
12 {
13     ClassA aobj=new ClassA();
14 }
```

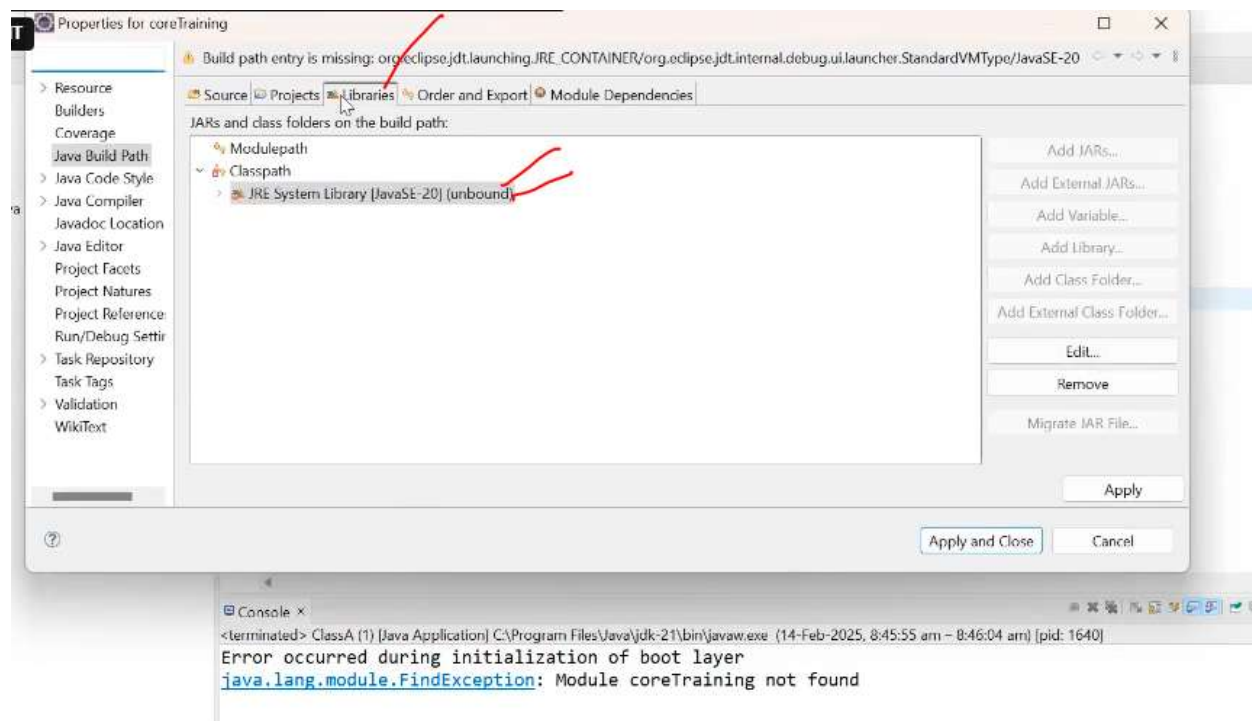
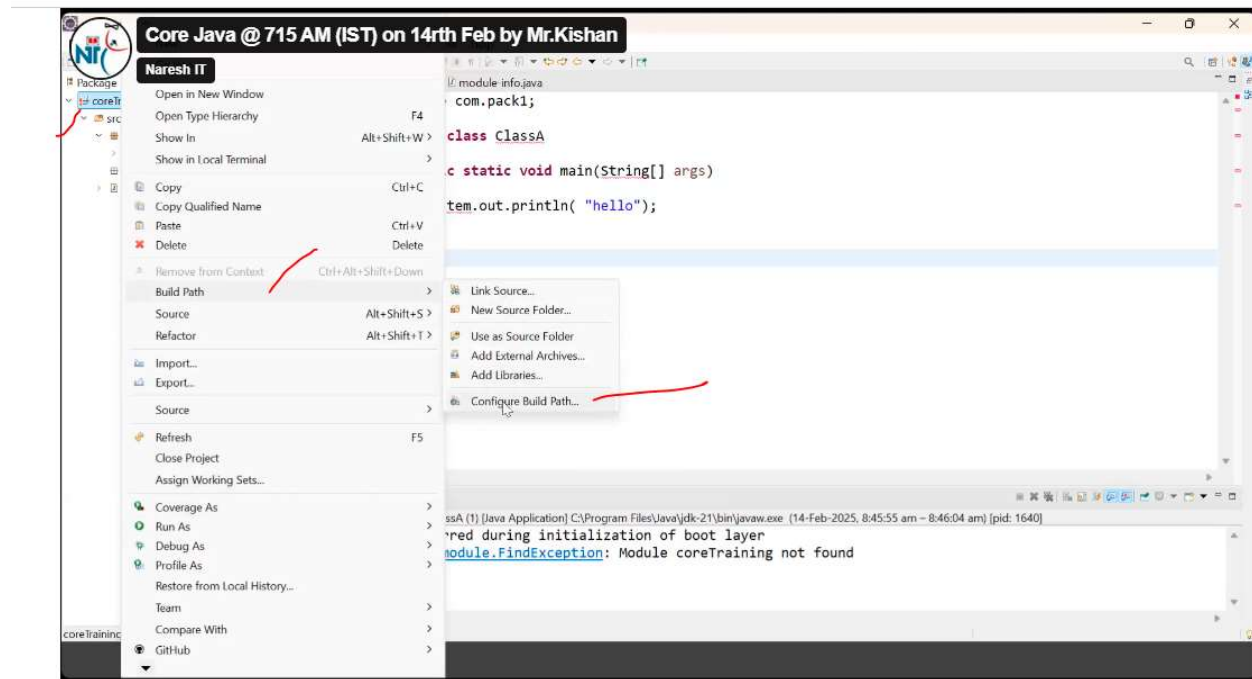
```
1 package com.pack2;
2
3 import com.pack1.ClassA;
4 import com.pack1.ClassB;
5
6 public class ClassX
7 {
8     public static void main(String[] args)
9     {
10         ClassA aobj=new ClassA();
11         aobj.meth1();
12
13         ClassB bobj=new ClassB();
14         bobj.meth2();
15     }
16 }
17
18
```

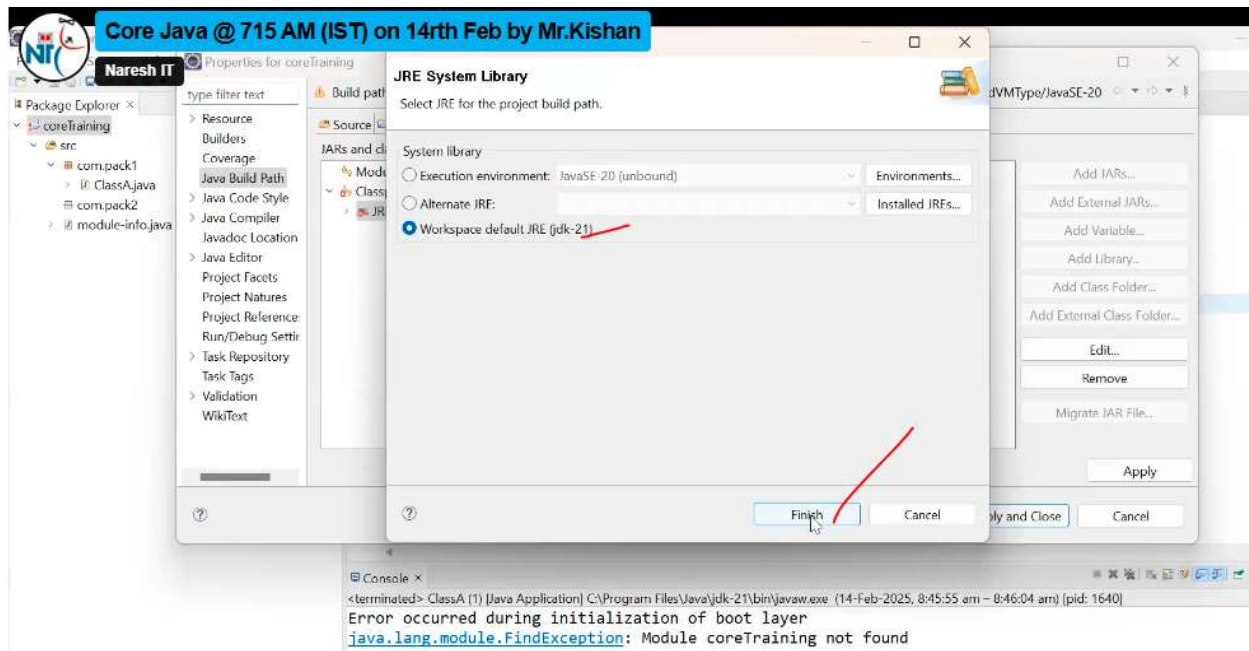
Console: <terminated> ClassX [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (14-Feb-2025, 8:30:05 am - 8:30:06 am)
Class A method

If you are getting below error kindly follow this

```
1 package com.pack1;
2
3 public class ClassA
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("hello");
8     }
9 }
10
```

Console: <terminated> ClassA (1) [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (14-Feb-2025, 8:45:55 am - 8:46:04 am) [pid: 1640]
Error occurred during initialization of boot layer
java.lang.module.FindException: Module coreTraining not found





Session 14 - 14th Feb

