

Summary of Common Java Basic Interview Questions (Part 1)

 [Guide](#)  Java  Java Basics  About 13913 words  About 46 minutes

This is a small advertisement that may be useful to you

- **Interview special edition** : Friends who are preparing for Java interviews can consider the interview special edition: "[Java Interview Guide](#)" (very high quality, specially designed for interviews, best used with JavaGuide).
- **Knowledge Planet** : Technical Column/One-on-one Questions/Resume Modification/Job Hunting Guide/Interview Check-in/Irregular Benefits, welcome to join [the JavaGuide official Knowledge Planet](#) .

Basic concepts and common sense

What are the characteristics of the Java language?

1. Easy to learn (simple syntax, easy to get started);
2. Object-oriented (encapsulation, inheritance, polymorphism);
3. Platform independence (Java virtual machine achieves platform independence);
4. Support for multithreading (C++ language does not have a built-in multithreading mechanism, so it is necessary to call the multithreading function of the operating system to perform multithreaded programming, while Java language provides multithreading support);
5. Reliability (with exception handling and automatic memory management mechanisms);
6. Security (the Java language itself is designed to provide multiple security protection mechanisms such as access rights modifiers and restrictions on direct program access to operating system resources);
7. Efficiency (through optimization technologies such as the Just In Time compiler, the Java language has very good operating efficiency);
8. Supports network programming and is very convenient;
9. Compilation and interpretation coexist;
10. ...





Fix (see: [issue#544](#)) : Starting with C++11 (in 2011), C++ introduced a multithreading library, which can be used to create threads on Windows, Linux, and macOS `std::thread` . `std::async` Reference link:
<http://www.cplusplus.com/reference/thread/thread/?kw=thread>



Expand a little:

The slogan "Write Once, Run Anywhere" is a true classic, having endured for years! Even today, many people still believe that Java's cross-platform nature is its greatest strength. In reality, cross-platform is no longer Java's biggest selling point, nor are the various new JDK features. Virtualization technology is now highly mature; for example, Docker makes cross-platform deployment easy. In my opinion, Java's powerful ecosystem is truly its most important selling point!

Java SE vs Java EE

- **Java SE (Java Platform, Standard Edition):** The Java Platform, Standard Edition, is the foundation of the Java programming language. It includes core components such as the core class libraries and virtual machine that support the development and execution of Java applications. Java SE can be used to build desktop applications or simple server applications.
- **Java EE (Java Platform, Enterprise Edition):** Built on Java SE, Java Platform, Enterprise Edition includes standards and specifications that support enterprise-level application development and deployment (such as Servlets, JSP, EJB, JDBC, JPA, JTA, JavaMail, and JMS). Java EE can be used to build distributed, portable, robust, scalable, and secure server-side Java applications, such as web applications.

Simply put, Java SE is the basic version of Java, and Java EE is the advanced version of Java. Java SE is more suitable for developing desktop applications or simple server applications, while Java EE is more suitable for developing complex enterprise applications or web applications.

In addition to Java SE and Java EE, there's also Java ME (Java Platform, Micro Edition). Java ME is a micro version of Java, primarily used for developing applications for embedded consumer electronics devices such as mobile phones, PDAs, set-top boxes, refrigerators, and air conditioners. Java ME isn't a major concern; just knowing about it is helpful; it's no longer used.



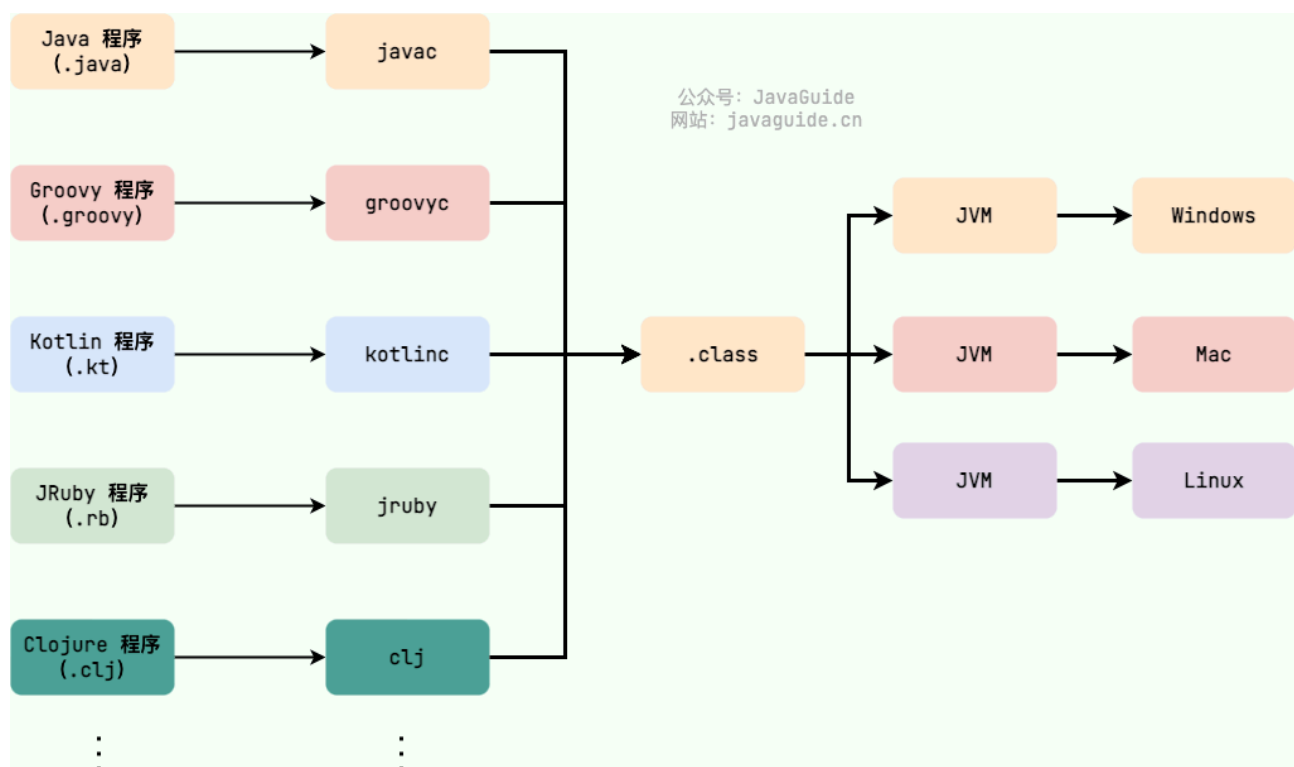
★ JVM vs JDK vs JRE

JVM

The Java Virtual Machine (JVM) is a virtual machine that runs Java bytecode. JVMs have specific implementations for different operating systems (Windows, Linux, and macOS). The goal is to produce the same results using the same bytecode. The bytecode and the different JVM implementations are key to the Java language's "compile once, run anywhere" philosophy.

As shown in the figure below, different programming languages (Java, Groovy, Kotlin, JRuby, Clojure, etc.) are compiled into files through their respective compilers

.class and ultimately run on different platforms (Windows, Mac, Linux) through the JVM.



There's more than just one JVM! Any company, organization, or individual can develop their own proprietary JVM, as long as it meets the JVM specification. In other words, the HotSpot VM we commonly encounter is just one implementation of the JVM specification.



HotSpot VM J9 VM Zing VM JRockit VM JVM
JVM Comparison of Java virtual machines
Java SE Specifications JDK JVM



Java SE > Java SE Specifications

Java Language and Virtual Machine Specifications

Java SE 18

Released March 2022 as JSR 393



The Java Language Specification, Java SE 18 Edition

- [HTML](#) | [PDF](#)
- Preview feature: [Pattern Matching for switch](#)



The Java Virtual Machine Specification, Java SE 18 Edition

- [HTML](#) | [PDF](#)

Java SE 17

Released September 2021 as JSR 392



The Java Language Specification, Java SE 17 Edition

- [HTML](#) | [PDF](#)
- Preview feature: [Pattern Matching for switch](#)



The Java Virtual Machine Specification, Java SE 17 Edition

- [HTML](#) | [PDF](#)

JDK JRE

JDK Java Development Kit

Java

Java

JRE

Java Runtime Environment

javac

javadoc

jdb

jconsole

javap

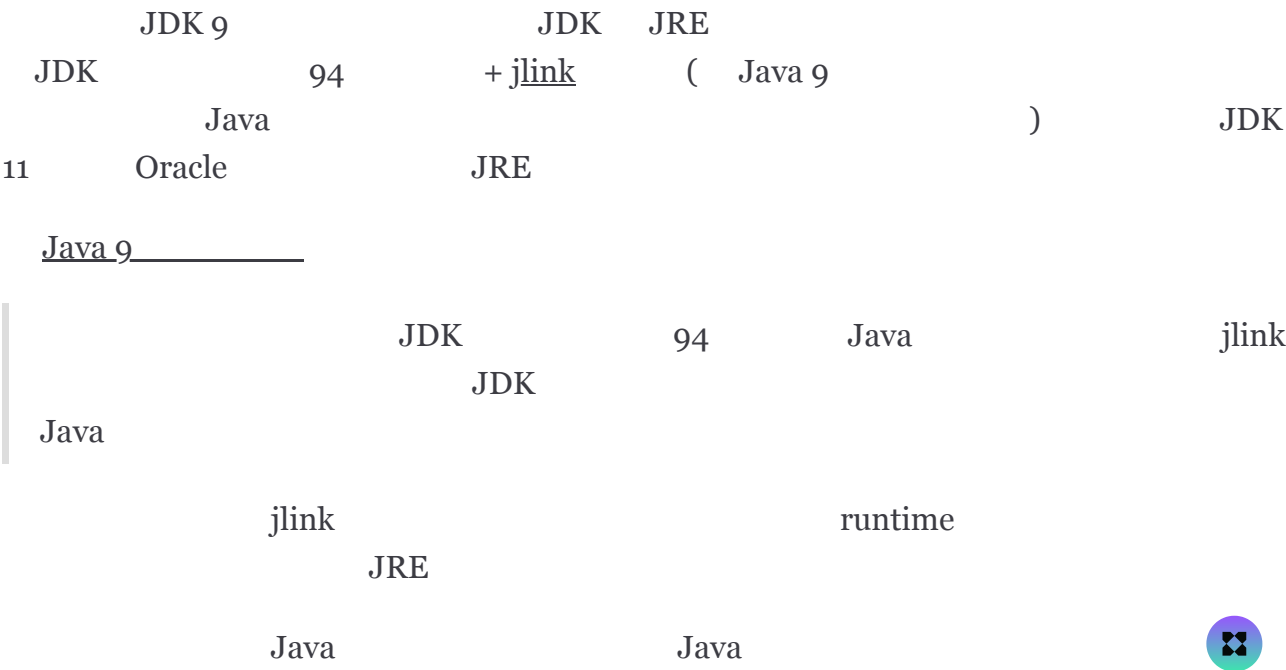
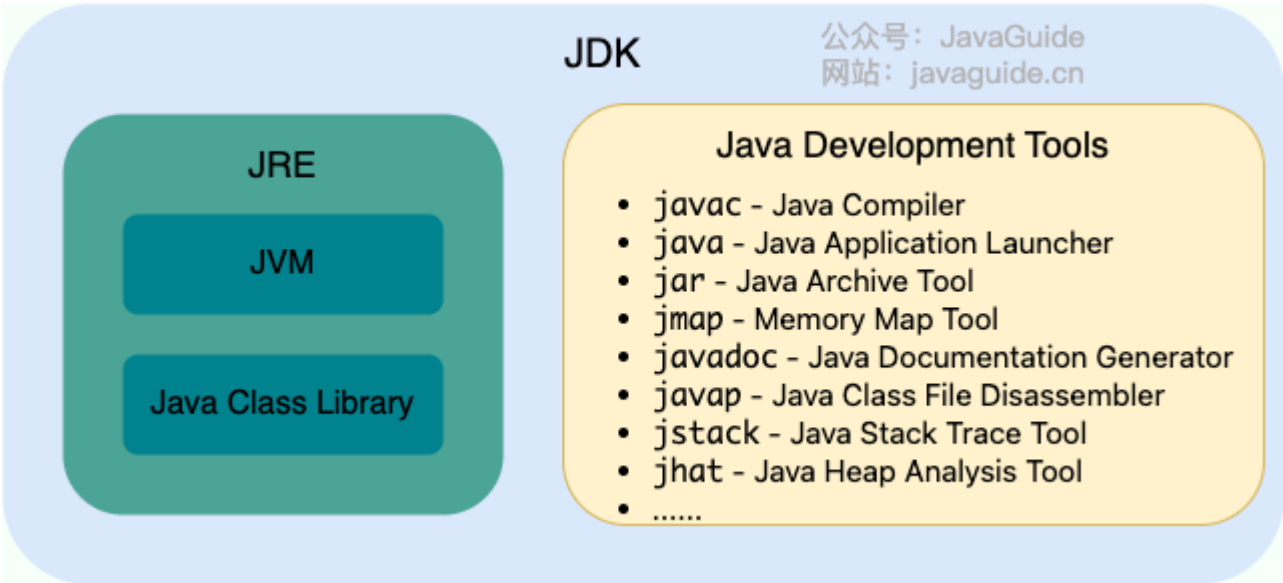
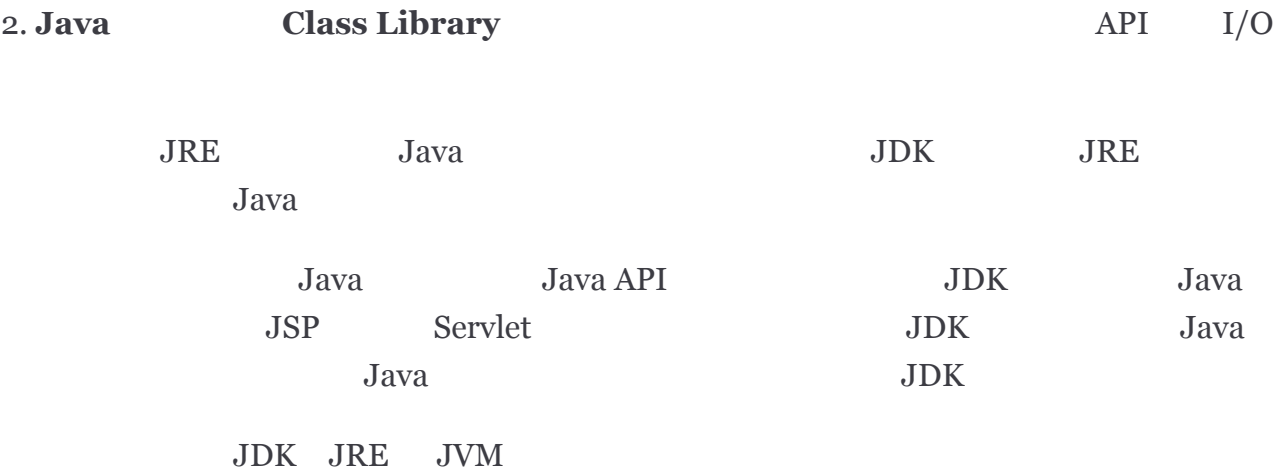
JRE

Java



1. JVM :

Java





?

?

Java

JVM

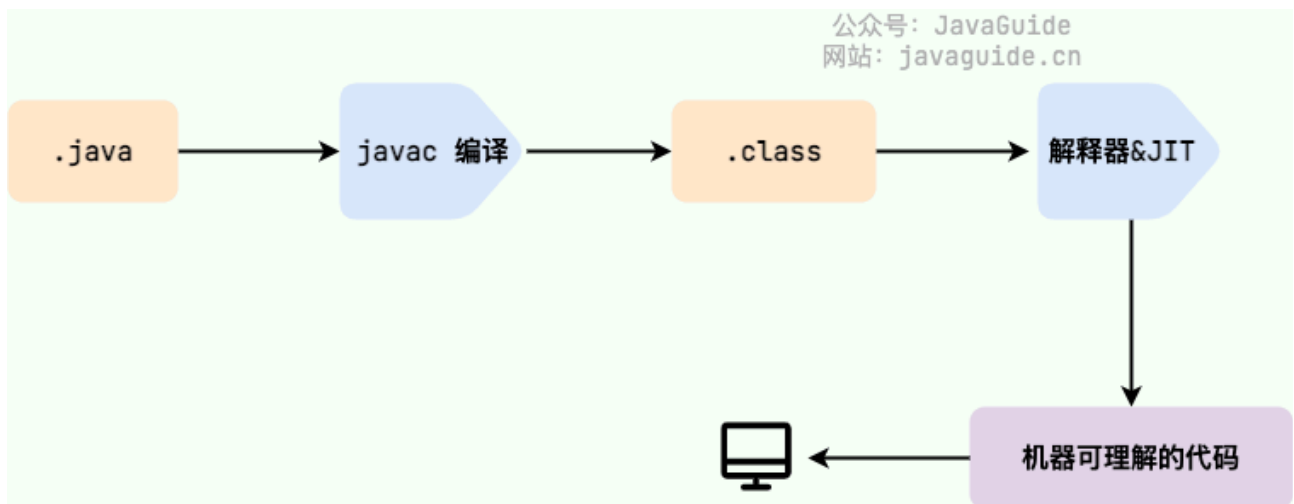
Java

.class

Java

C C++ Rust Go
Java

Java



.class->

JVM

()

JIT Just

in Time Compilation

JIT

JIT

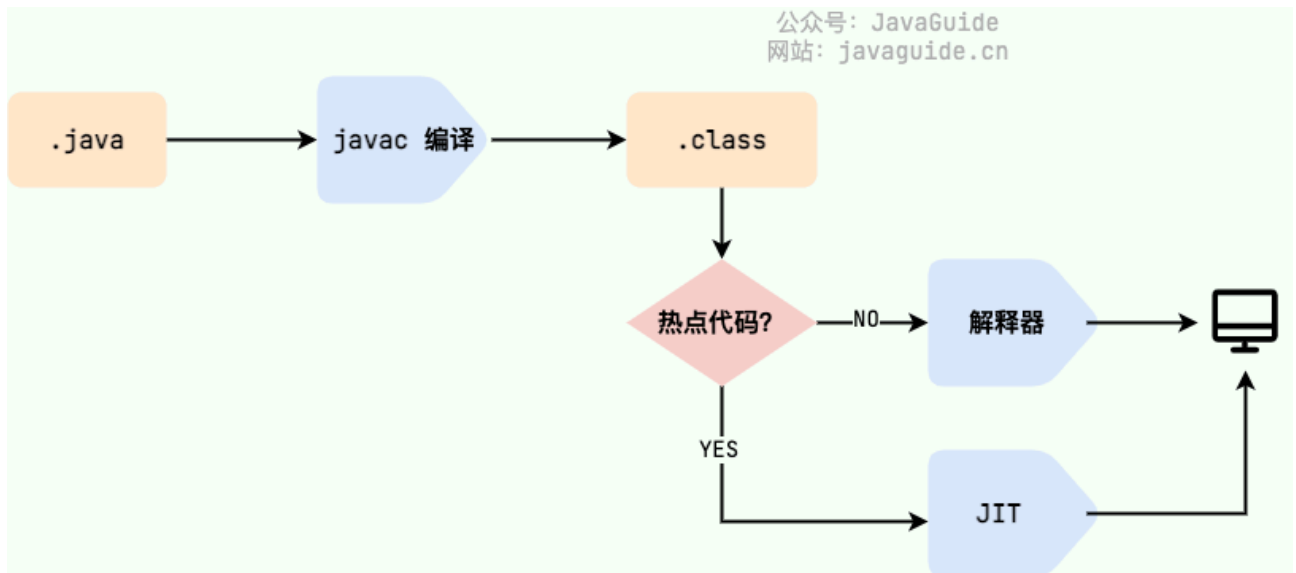
Java

Java



- _____ | Java _____ - _____
- _____ - _____





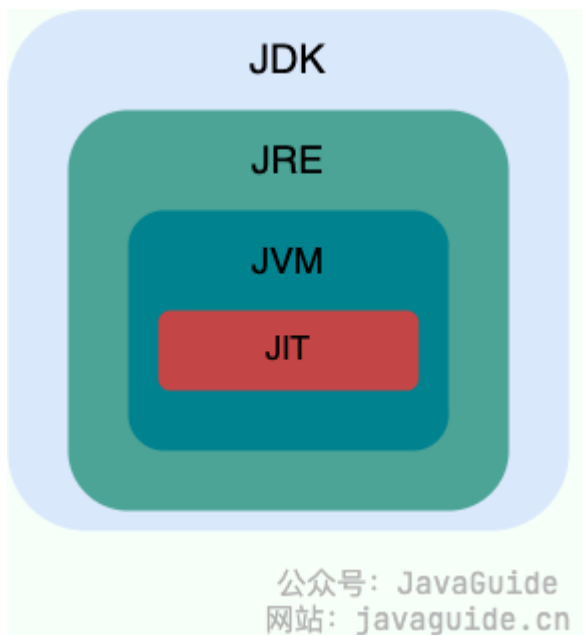
HotSpot

(Lazy Evaluation)

JIT

JVM

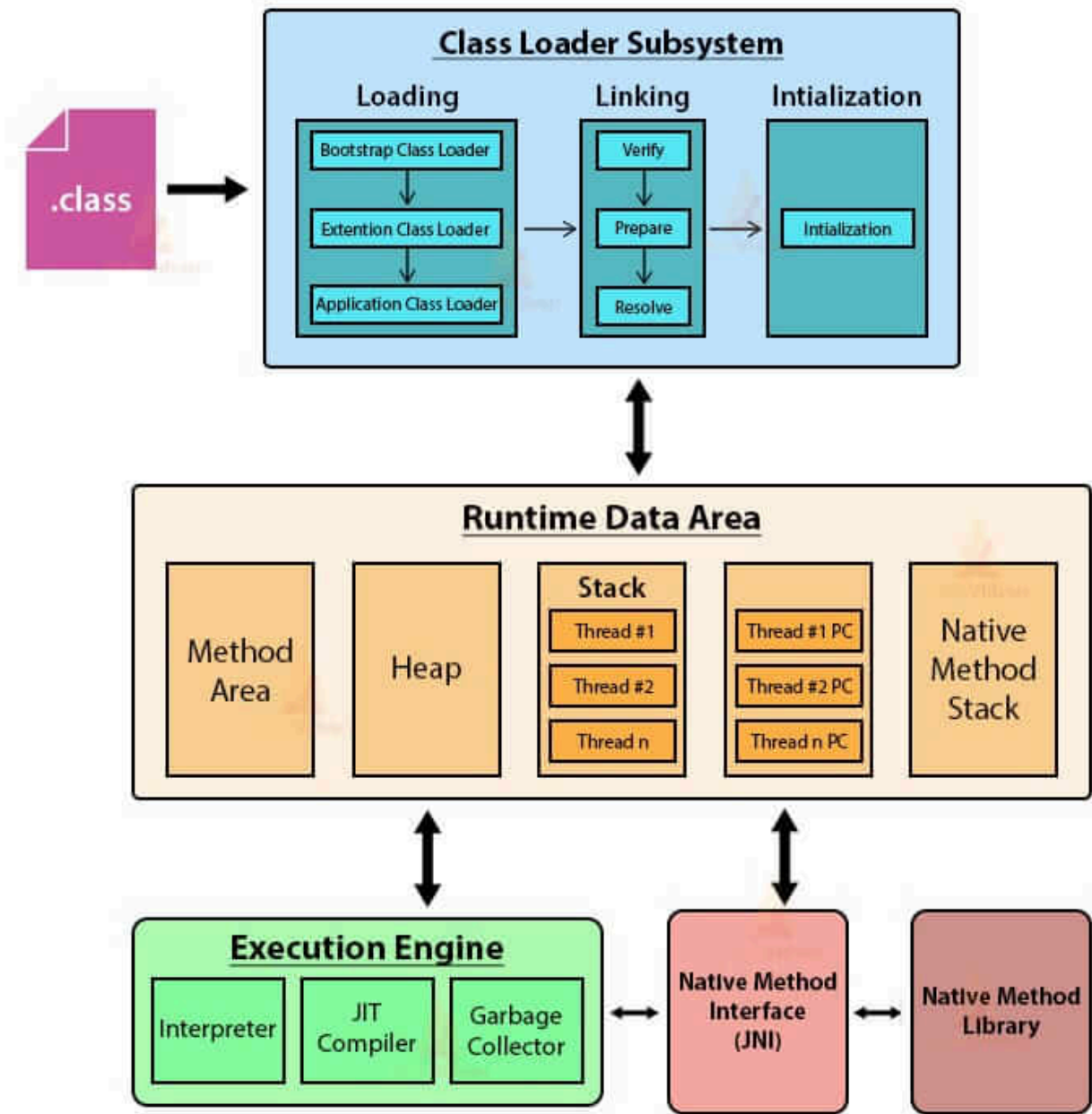
JDK JRE JVM JIT



JVM



JVM Model



Java

“

”

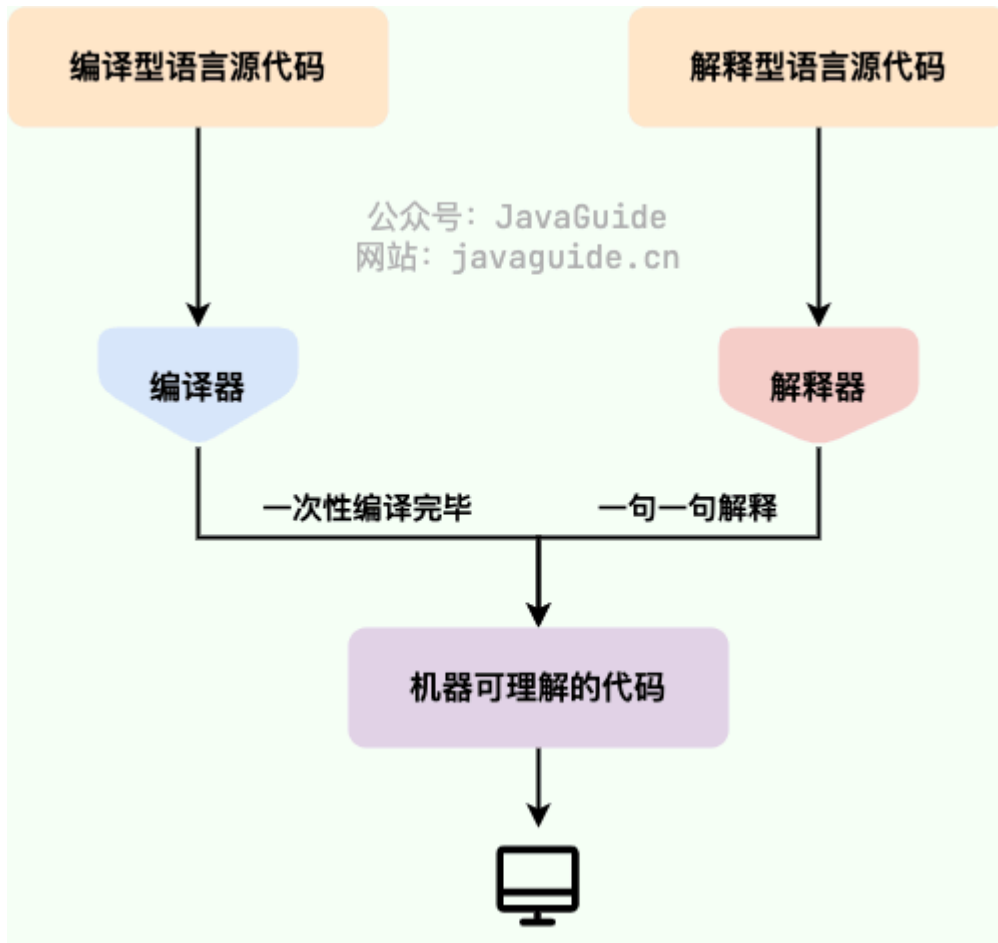
•

C C++ Go Rust



interpret

Python JavaScript PHP

[illegible]

Java

Java “ ”

Java

Java

Java

```
.class
```

Java



AOT

AOT

JDK 9

AOT(Ahead of Time Compilation)

JIT

C C++

Rust Go

AOT

JIT

Java

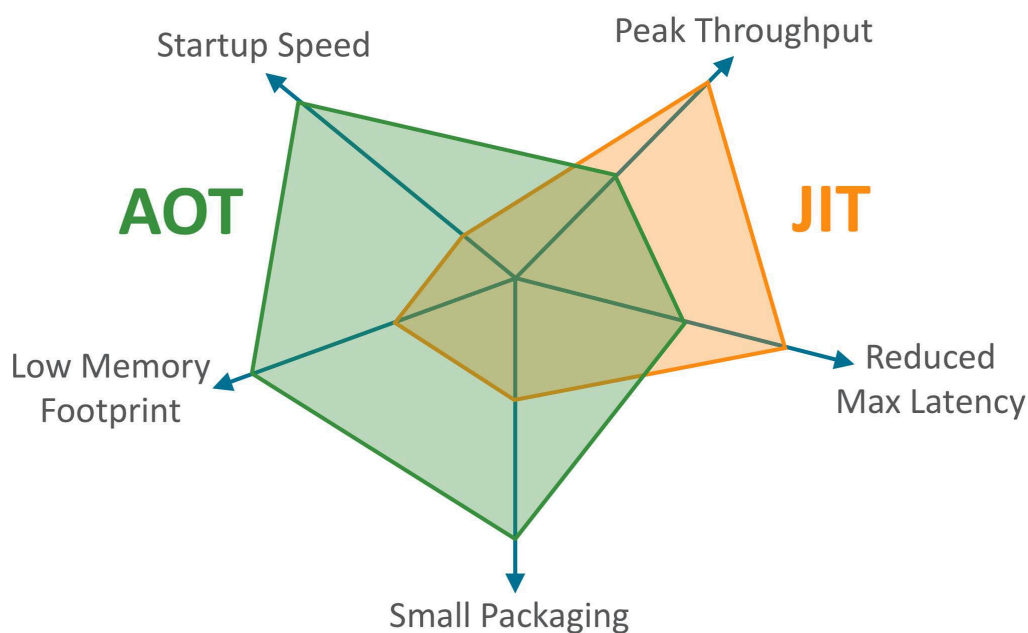
AOT

Java

AOT

JIT **AOT**

:



AOT

JIT

AOT

GraalVM

GraalVM

JDK

JDK

Java

JVM

JavaScript

Python

JVM

GraalVM

AOT

JIT

GraalVM

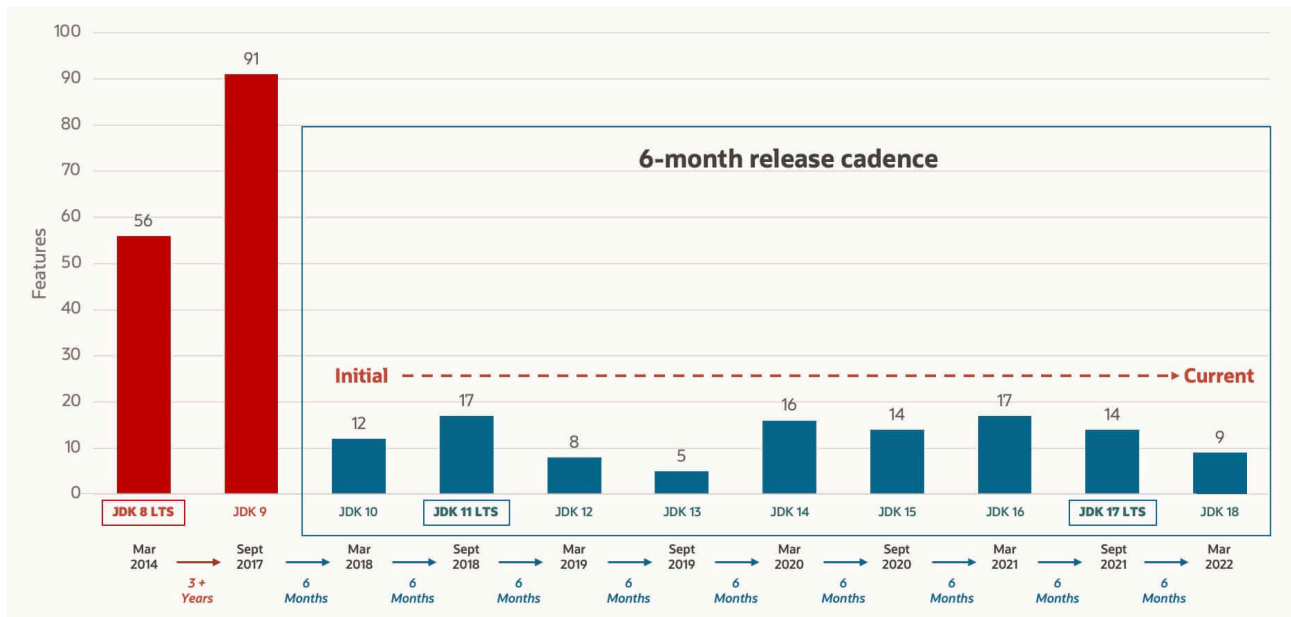
<https://www.graalvm.org/latest/docs/>

- _____
- Native Spring&Dubbo AOT

AOT







Oracle JDK OpenJDK

OpenJDK
Alibaba Dragonwell

OpenJDK

AWS

Amazon Corretto



- BCL Oracle Binary Code License Agreement JDK
- OTN Oracle Technology Network License Agreement 11 JDK

Java C++ ?

C++
C++

Java C++

Java C++

- Java
- Java C++ Java
- Java (GC)
- C++ Java
- Java
-



Java

1.

2.

3. Java

```
/**
 * Return whether this cache manager stores a copy of each entry or
 * a reference for all its caches. If store by value is enabled, any
 * cache entry must be serializable.
 * @since 4.3
 */
public boolean isStoreByValue() {
    return this.storeByValue;
}

@Override
public void setBeanClassLoader(ClassLoader classLoader) {
    this.serialization = new SerializationDelegate(classLoader);
    // Need to recreate all Cache instances with new ClassLoader in store-by-value mode...
    if (isStoreByValue()) {
        recreateCaches();
    }
}
```

文档注释

单行注释

Clean Code



1

// check to see if the employee is eligible for full benefits java

2

if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))

1

if (employee.isEligibleForFullBenefits())

java


Java

“ ”

“ ” “ ”

“ ”

Java

	private	protected	public			
	abstract	class	extends	final	implements	inter
	new	static	strictfp	synchronized	transient	volat
	break	continue	return	do	while	 11

	for	instanceof	switch	case	default	asse
	try	catch	throw	throws	finally	
	import	package				
	boolean	byte	char	double	float	int
	short					
	super	this	void			
	goto	const				

Tips
default

IDE

- switch default
- JDK8 default
- default



true , false , null

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>



1 1 Java

(++) (--)

++ --

- ++a --a /
b = ++a a 1 b
 - a++ a-- /
b = a++ a b a 1
- / /
- a b c
- d e

```

1  int a = 9;
2  int b = a++;
3  int c = ++a;
4  int d = c--;
5  int e = --d;

```

a = 11 b = 9 c = 10 d = 10 e = 10



JDK

HashMap JDK1.8

hash

```

1  static final int hash(Object key) {
2      int h;
3      // key.hashCode()                      hashCode
4      // ^
5      // >>>:                      0
6      return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
7  }

```

1.



2.

int long

2

-
-
-

- CRC
-

Java

- \ll : $x \ll n$, $x \cdot 2^n$
()
- \gg : $x \gg n$, $x \cdot 2^n$
1 $x \gg n$, $x \cdot 2^n$
0
- \ggg :

double float

int long short byte char
int

int / 32 % /
/ 10 42%32=10 long 32%32=0 / 42
64 64 long
x<<42 x<<10 x>>42 x>>10 x >>>42 x >>>

10



```

1  int i = -1;
2  System.out.println("      " + i);
3  System.out.println("      " +
4  Integer.toBinaryString(i));
5  i <= 10;
6  System.out.println("      10      " + i);
   System.out.println("      10      " +
   Integer.toBinaryString(i));

```

java

```

1      -1
2      11111111111111111111111111111111
3      10      -1024
4      10      111111111111111111111111100000000000

```

plain

42 32 %
 10 42%32=10

```

1  int i = -1;
2  System.out.println("      " + i);
3  System.out.println("      " +
4  Integer.toBinaryString(i));
5  i <= 42;
6  System.out.println("      10      " + i);
   System.out.println("      10      " +
   Integer.toBinaryString(i));

```

java

continue break return

1. continue
2. break

return

return



1. return; return
2. return value; return

```
1 public static void main(String[] args) {
2     boolean flag = false;
3     for (int i = 0; i <= 3; i++) {
4         if (i == 0) {
5             System.out.println("0");
6         } else if (i == 1) {
7             System.out.println("1");
8             continue;
9         } else if (i == 2) {
10            System.out.println("2");
11            flag = true;
12        } else if (i == 3) {
13            System.out.println("3");
14            break;
15        } else if (i == 4) {
16            System.out.println("4");
17        }
18        System.out.println("xixi");
19    }
20    if (flag) {
21        System.out.println("haha");
22        return;
23    }
24    System.out.println("heihei");
25 }
```

java

```
1 0
2 xixi
3 1
4 2
5 xixi
6 3
7 haha
```

plain



Java

Java 8

- 6
 - 4 byte short int long
 - 2 float double
- 1 char
- 1 boolean

8

byte	8	1	0	-128 ~ 127
short	16	2	0	-32768 ~ 32767
int	32	4	0	-2147483648 ~ 2147483647
long	64	8	0L	-9223372036854775808 ~ 9223372036854775807
char	16	2	'u0000'	0 ~ 65535 (2 ¹⁶ - 1)
float	32	4	of	1.4E-45 ~ 3.4028235E38
double	64	8	od	4.9E-324 ~ 1.7976931348623157E308
boolean	1		false	true, false

You can see that the maximum representable positive numbers, such as byte , short , , int and , long are all reduced by 1. Why is this? In two's complement notation, the top bit is used to indicate the sign (0 for positive numbers, 1 for negative numbers), while



remaining bits represent the magnitude. Therefore, to represent the maximum positive number, we need to set all bits except the top bit to 1. If we add 1, we will overflow, resulting in a negative number.

The official documentation does not clearly define this `boolean` ; it depends on the specific implementation by the JVM vendor. Logically, it occupies 1 bit, but in practice, efficient computer storage is considered.

Additionally, the memory size occupied by each Java primitive type does not change with changes in machine hardware architecture, as it does in most other languages. This invariance is one of the reasons why Java programs are more portable than programs written in most other languages (as mentioned in Section 2.2 of "Thinking in Java").

Notice:

1. When using data of type in Java , you must add **L** `long` after the value , otherwise it will be parsed as an integer.
2. When using data of type in Java, you must add **f** or **F** `float` after the value , otherwise it will not compile.
3. `char a = 'h'` char: single quote, `String a = "hello"` : double quote.

These eight basic types have corresponding packaging classes: `Byte` , `Short` , `Integer` , `Long` , `Float` , `Double` , `Character` , `Boolean` .

What is the difference between primitive types and package types?

- **Usage** : Apart from defining some constants and local variables, we rarely use primitive types to define variables in other places, such as method parameters and object properties. In addition, wrapper types can be used in generics, while primitive types cannot.
- **Storage** : Local variables of primitive data types are stored in the local variable table on the Java Virtual Machine stack. Member variables of primitive data types (unmodified `static`) are stored in the Java Virtual Machine's heap. Wrapper types are object types, and we know that almost all object instances exist on the heap.
- **Space occupied** : Compared with packaging types (object types), the space occupied by basic data types is often very small.
- **Default value** : The member variable wrapper type is not assigned a value `null` , while the basic type has a default value and is not `null` .
- **Comparison method** : For primitive data types, `==` the comparison is based on the value. For wrapper data types, `==` the comparison is based on the memory address of



the object. All integer wrapper objects are compared using the comparison `equals()` method.

Why do we say that almost all object instances exist on the heap? This is because after the HotSpot virtual machine introduced JIT optimization, it will perform escape analysis on the object. If it finds that an object has not escaped outside the method, it may be possible to achieve stack allocation through scalar replacement, thereby avoiding heap allocation memory.

⚠ Note: It's a common misconception that primitive types are stored on the stack! The storage location of primitive types depends on their scope and declaration. If they are local variables, they are stored on the stack; if they are member variables, they are stored in the heap/method area/metaspase.

```

1  public class Test {
2      //
3      int a = 10;
4      // static          JDK 1.7          1.8
5
6      //
7      static int b = 20;
8
9      public void method() {
10         //
11         int c = 30;
12         static int d = 40; //          static
13     }
14 }

```

Do you understand the caching mechanism of packaging types?

Most of the wrapper types of Java basic data types use caching mechanisms to improve performance.

Byte , Short , Integer , Long These 4 wrapper classes create cache data of the corresponding type with values **[-128, 127]** Character by default, create cache data with values in the range of **[0, 127]** Boolean , and directly return TRUE or FALSE .



You can modify the cache upper limit through Integer JVM parameters - `XX:AutoBoxCacheMax=<size>` , but you cannot modify the lower limit -128. In actual use, it is not recommended to set too large a value to avoid wasting memory or even OOM.

For Byte , Short , Long , Character there is no similar `-XX:AutoBoxCacheMax` parameter that can be modified, so the cache range is fixed and cannot be adjusted through JVM parameters. Boolean directly returns the predefined `TRUE` and `FALSE` instances, without the concept of cache range.

Integer cache source code:

```
1 public static Integer valueOf(int i) {
2     if (i >= IntegerCache.low && i <= IntegerCache.high)
3         return IntegerCache.cache[i + (-IntegerCache.low)];
4     return new Integer(i);
5 }
6 private static class IntegerCache {
7     static final int low = -128;
8     static final int high;
9     static {
10         // high value may be configured by property
11         int h = 127;
12     }
13 }
```

java

Character Cache source code:

```
1 public static Character valueOf(char c) {
2     if (c <= 127) { // must cache
3         return CharacterCache.cache[(int)c];
4     }
5     return new Character(c);
6 }
7
8 private static class CharacterCache {
9     private CharacterCache(){}
10    static final Character cache[] = new Character[127 + 1];
11    static {
12
13
14
15 }
```

java


```

16         for (int i = 0; i < cache.length; i++)
            cache[i] = new Character((char)i);
    }
}

```

Boolean Cache source code:

```

1    public static Boolean valueOf(boolean b) {
2        return (b ? TRUE : FALSE);
3    }

```

java

If the corresponding range is exceeded, a new object will still be created. The size of the cache range is just a trade-off between performance and resources.

The wrapper classes for the two floating-point number types `Float` do `Double` not implement a caching mechanism.

```

1    Integer i1 = 33;
2    Integer i2 = 33;
3    System.out.println(i1 == i2);//      true
4
5    Float i11 = 333f;
6    Float i22 = 333f;
7    System.out.println(i11 == i22);//      false
8
9    Double i3 = 1.2;
10   Double i4 = 1.2;
11   System.out.println(i3 == i4);//      false

```

java


Let's look at a question: What is the output of the following code `true` or `false` ?

```

1    Integer i1 = 40;
2    Integer i2 = new Integer(40);
3    System.out.println(i1==i2);

```

java

`Integer i1=40` This line of code will cause boxing, which means that this line of code is equivalent to `Integer i1=Integer.valueOf(40)` . Therefore, `i1` the object in the cache is used directly. `Integer i2 = new Integer(40)` will directly create a new object. 

So the answer is `false` . Did you get it right?

Remember: **All comparisons between integer wrapper class objects use the equals method .**

7. **【强制】**所有整型包装类对象之间值的比较，全部使用 equals 方法比较。

说明：对于 Integer var = ? 在-128 至 127 之间的赋值，Integer 对象是在 IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用 == 进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用 equals 方法进行判断。

Do you know about automatic boxing and unboxing? What is the principle?

What is automatic unpacking?

- **Boxing** : Wrapping primitive types with their corresponding reference types;
- **Unboxing** : converting the package type to the basic data type;

For example:

```
1 Integer i = 10; //
2 int n = i; //
```

java

The bytecode corresponding to the above two lines of code is:

```
1 L1
2
3 LINENUMBER 8 L1
4
5 ALOAD 0
6
7 BIPUSH 10
8
9 INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
10
11 PUTFIELD AutoBoxTest.i : Ljava/lang/Integer;
12
13 L2
14
15 LINENUMBER 9 L2
16
```

java



```

17      ALOAD 0
18
19      ALOAD 0
20
21      GETFIELD AutoBoxTest.i : Ljava/lang/Integer;
22
23      INVOKEVIRTUAL java/lang/Integer.intValue ()I
24
25      PUTFIELD AutoBoxTest.n : I
26
27      RETURN

```

From the bytecode, we find that boxing is actually calling `valueOf()` the method of the packaging class, and unboxing is actually calling `xxxValue()` the method.

therefore,

- `Integer i = 10` Equivalent to `Integer i = Integer.valueOf(10)`
- `int n = i` is equivalent to `int n = i.intValue()` ;

Note: Frequent unpacking and packing will seriously affect system performance. We should try to avoid unnecessary unpacking and packing.

```

1      private static long sum() {
2          //          long          Long
3          Long sum = 0L;
4          for (long i = 0; i <= Integer.MAX_VALUE; i++)
5              sum += i;
6          return sum;
7      }

```

java

Why is there a risk of loss of precision when doing floating-point operations?

Floating point operation precision loss code demonstration:

```

1      float a = 2.0f - 1.9f;
2      float b = 1.8f - 1.7f;
3      System.out.printf("%.9f",a);// 0.100000024
4      System.out.println(b);// 0.0999999905
5      System.out.println(a == b);// false

```

java

Why does this problem occur?

This has a lot to do with how computers store floating-point numbers. Computers use binary, and the width of a number is limited. Infinitely recurring decimals are truncated when stored in a computer, resulting in a loss of decimal precision. This explains why floating-point numbers cannot be accurately represented in binary.

For example, the decimal number 0.2 cannot be accurately converted into a binary decimal:


```
1 // 0.2                                     2                                     java
2 //
3 0.2 * 2 = 0.4 -> 0
4 0.4 * 2 = 0.8 -> 0
5 0.8 * 2 = 1.6 -> 1
6 0.6 * 2 = 1.2 -> 1
7 0.2 * 2 = 0.4 -> 0
8 ...
```

For more information about floating-point numbers, I recommend reading the article [Computer System Fundamentals \(IV\) Floating-Point Numbers](#).

How to solve the problem of precision loss in floating-point operations?

`BigDecimal` It can perform operations on floating-point numbers without losing precision. Generally speaking, most business scenarios that require accurate floating-point calculation results (such as those involving money) are `BigDecimal` performed through.

```
1 BigDecimal a = new BigDecimal("1.0");
2 BigDecimal b = new BigDecimal("1.00");
3 BigDecimal c = new BigDecimal("0.8");
4
5 BigDecimal x = a.subtract(c);
6 BigDecimal y = b.subtract(c);
7
8 System.out.println(x); /* 0.2 */
9
10
11
12
```



```

13  System.out.println(y); /* 0.20 */
    //
    System.out.println(Objects.equals(x, y)); /* false */
    //          compareTo          0
    System.out.println(0 == x.compareTo(y)); /* true */

```

For `BigDecimal` a detailed introduction to `BigDecimal`, you can read this article I wrote: [Detailed Explanation of BigDecimal](#) .

How should data exceeding the long integer type be represented?

Basic numeric types have an expression range. If the range is exceeded, there is a risk of numerical overflow.

In Java, the 64-bit long integer is the largest integer type.

```

1  long l = Long.MAX_VALUE;
2  System.out.println(l + 1); // -9223372036854775808
3  System.out.println(l + 1 == Long.MIN_VALUE); // true

```

java

`BigInteger` Arrays are used internally `int[]` to store integer data of any size.

Compared with regular integer type operations, `BigInteger` the efficiency of the operation will be relatively low.

variable

★ What is the difference between member variables and local variables?

- **Syntax form** : From the grammatical form, member variables belong to the class, while local variables are variables defined in the code block or method or are parameters of the method; member variables can be modified by modifiers such as `public` , `private` , `static` etc., while local variables cannot be modified by access control modifiers and `static` ; however, both member variables and local variables can be `final` modified by .
- **Storage method** : From the perspective of variable storage in memory, if a member variable is `static` modified with , it belongs to the class. If it is not `static` modified



with , it belongs to the instance. Objects are stored in heap memory, while local variables are stored in stack memory.

- **Survival time** : From the perspective of the survival time of variables in memory, member variables are part of the object and exist when the object is created, while local variables are automatically generated when the method is called and disappear when the method call ends.
- **Default value** : From the perspective of whether the variable has a default value, if the member variable is not assigned an initial value, it will be automatically assigned the default value of the type (with one exception: `final` the modified member variable must also be explicitly assigned a value), while local variables will not be automatically assigned a value.

Why do member variables have default values?

1. Regardless of the variable type, what happens if there is no default value? The variable stores an arbitrary random value corresponding to the memory address, and the program will encounter unexpected results when reading this value.
2. There are two ways to set default values: manual and automatic. According to the first point, if there is no manual assignment, it must be automatically assigned. Member variables can be manually assigned at runtime using methods such as reflection, but local variables cannot.
3. For the compiler (javac), it's easy to tell if a local variable has no value assigned, and a direct error will be reported. However, member variables may be assigned values at runtime, making this difficult to determine. A false "no default value" report would affect the user experience, so automatic default value assignment is used.

Member variables and local variables code example:

```
1 public class VariableExample {
2
3     //
4     private String name;
5     private int age;
6
7     //
8     public void method() {
9         int num1 = 10; //
10        String str = "Hello, world!"; //
11        System.out.println(num1);
12        System.out.println(str);
13    }
14 }
```

java

```
13     }
14
15     //
16     public void method2(int num2) {
17         int sum = num2 + 10; //
18         System.out.println(sum);
19     }
20
21     //
22     public VariableExample(String name, int age) {
23         this.name = name; //
24         this.age = age; //
25         int num3 = 20; //
26         String str2 = "Hello, " + this.name + "!"; //
27
28         System.out.println(num3);
29         System.out.println(str2);
30     }
}
```

What is the function of static variables?

A static variable is `static` a variable modified by the keyword. It is shared by all instances of a class. Regardless of how many objects are created within a class, they all share the same copy of the static variable. This means that static variables are allocated memory only once, even if multiple objects are created, saving memory.

Static variables are accessed by the class name, for example

`StaticVariableExample.staticVar` (`private` this is not possible if they are qualified with a keyword).

```
1     public class StaticVariableExample {
2         //
3         public static int staticVar = 0;
4     }
```

java

Typically, static variables are `final` modified by keywords to become constants.



```
1 public class ConstantVariableExample {
2     //
3     public static final int constantVar = 0;
4 }
```

java

What is the difference between character constants and string constants?

- **Format** : A character constant is a single character enclosed in single quotes, and a string constant is zero or more characters enclosed in double quotes.
- **Meaning** : A character constant is equivalent to an integer value (ASCII value) and can participate in expression operations; a string constant represents an address value (the location where the string is stored in memory).
- **Memory size** : Character constants only take up 2 bytes; string constants take up several bytes.

⚠ Note: char occupies two bytes in Java.

Code examples for character constants and string constants:

```
1 public class StringExample {
2     //
3     public static final char LETTER_A = 'A';
4
5     //
6     public static final String GREETING_MESSAGE = "Hello, world!";
7     public static void main(String[] args) {
8         System.out.println("
9         "+Character.BYTES);
10        System.out.println("
11        "+GREETING_MESSAGE.getBytes().length);
12    }
13 }
```

java

Output:

```
1          2
2          13
```

plain

method

What is the return value of a method? What types of methods are there?

The return value of a method is the result of executing the code in the method body (assuming the method can produce a result). The purpose of the return value is to receive the result so that it can be used in other operations.

We can classify methods into the following types according to their return value and parameter types:

1. Methods with no parameters and no return value

```
1  public void f1() {
2      //.....
3  }
4  //                                return
5  public void f(int a) {
6      if (...) {
7          //                        ,
8          return;
9      }
10     System.out.println(a);
11 }
```

java


2. Methods with parameters but no return value

```
1  public void f2(Parameter 1, ..., Parameter n) {
2      //.....
3  }
```

java

3. Methods with return values and no parameters

```
1  public int f3() {
2      //.....
3      return x;
4  }
```



4. Methods with return values and parameters

```
1 public int f4(int a, int b) {  
2     return a * b;  
3 }
```

java

Why can't static methods call non-static members?

This requires the combination of JVM related knowledge, the main reasons are as follows:

1. Static methods belong to the class, and memory is allocated when the class is loaded. They can be accessed directly through the class name. Non-static members, on the other hand, belong to instance objects and only exist after the object is instantiated. They need to be accessed through the instance object of the class.
2. The static method already exists when the non-static members of the class do not exist. At this time, calling the non-static member that does not exist in the memory is an illegal operation.

```
1 public class Example {  
2     //  
3     public static final char LETTER_A = 'A';  
4  
5     //  
6     public static final String GREETING_MESSAGE = "Hello, world!";  
7  
8     public static void main(String[] args) {  
9         //  
10        System.out.println("        " + LETTER_A);  
11  
12        //  
13        System.out.println("        " + GREETING_MESSAGE);  
14    }  
15 }
```

java

★ What is the difference between static methods and instance methods?

1. Calling method



When calling static methods externally, you can use `ClassName.method()` either the `__name__` or `__name__.method()` the `__name__` method, while instance methods can only use the `__name__` method. In other words, **calling static methods does not require creating an object**.

However, it should be noted that it is generally not recommended to use `ClassName.method()` the method of calling static methods. This method is very easy to cause confusion. Static methods do not belong to a certain object of the class but to the class itself.

Therefore, it is generally recommended to use `obj.method()` the method to call static methods.

```
1 public class Person {
2     public void method() {
3         //.....
4     }
5
6     public static void staicMethod(){
7         //.....
8     }
9     public static void main(String[] args) {
10         Person person = new Person();
11         //
12         person.method();
13         //
14         Person.staicMethod()
15     }
16 }
```

java

2. Are there any restrictions on accessing class members?

When static methods access members of this class, they are only allowed to access static members (i.e. static member variables and static methods), and are not allowed to access instance members (i.e. instance member variables and instance methods). This restriction does not exist for instance methods.

★ What is the difference between overloading and overriding?

Overloading means that the same method can be processed differently according to different input data.



Overriding is when a subclass inherits the same method from the parent class, has the same input data, but wants to respond differently from the parent class, you have to override the parent class method

Overload

Occurs in the same class (or between a parent class and a child class), the method name must be the same, the parameter types, numbers, and orders are different, and the method return values and access modifiers can be different.

The book "Java Core Technology" introduces overloading as follows:

If multiple methods (such as `StringBuilder` the constructor) have the same name and different parameters, overloading occurs.

```
1   StringBuilder sb = new StringBuilder();                                java
2   StringBuilder sb2 = new StringBuilder("HelloWorld");
```

The compiler must choose which method to execute. It does this by matching the argument types given by each method with the value types used in a particular method call. A compile-time error occurs if the compiler can't find a matching argument, either because there is no match or because no method is better than the others (this process is called overloading resolution).

Java allows overloading of any method, not just constructor methods.

To sum up: overloading means that multiple methods with the same name in the same class perform different logical processing according to different parameters.

Rewrite

Overriding occurs at runtime, and is the process of a subclass rewriting the implementation of the parent class's allowed access methods.

1. The method name and parameter list must be the same. The return value type of the subclass method should be smaller than or equal to the return value type of the parent class method. The scope of the thrown exception is smaller than or equal to the parent class, and the scope of the access modifier is greater than or equal to the parent class.
2. If the parent class method access modifier is `private`/`final`/`static` the subclass cannot override the method, but `static` the modified method can be declared again.
3. Constructors cannot be overridden



Summarize

To sum up: **overriding is the transformation of the parent class method by the subclass. The external appearance cannot be changed, but the internal logic can be changed.**

Differences	Overloading	Overriding
Occurrence range	In the same class.	There is an inheritance relationship between the parent class and the child class.
Method signature	The method names must be the same , but the parameter lists must be different (at least one of the parameter type, number, or order must be different).	The method name and parameter list must be exactly the same .
Return Type	It has nothing to do with the return value type and can be modified arbitrarily.	The return type of a subclass method must be the same as the return type of the superclass method , or a subclass of it .
Access modifiers	It has nothing to do with the access modifier and can be modified arbitrarily.	The access rights of a subclass method cannot be lower than those of the superclass method. (public > protected > default > private)
Binding period	Compile-time binding or static binding	Run-time Binding or Dynamic Binding

Method rewriting should follow the "two same, two small, and one large" (the following content is excerpted from "Crazy Java Lecture Notes", [issue#892](#)):



- "Two same" means the method name is the same and the parameter list is the same;

- "Two small" means that the return value type of the subclass method should be smaller or equal to the return value type of the parent class method, and the exception class declared thrown by the subclass method should be smaller or equal to the exception class declared thrown by the parent class method;
- "A large" means that the access rights of the subclass method should be greater than or equal to the access rights of the parent class method.

★ Regarding **overriding return types**: A few words of clarification are needed here, as the above description isn't entirely clear: if a method's return type is void or a primitive data type, the return value cannot be modified when overriding. However, if the method's return value is a reference type, a subclass of that reference type can be returned when overriding.

```
1  public class Hero {
2      public String name() {
3          return " ";
4      }
5  }
6  public class SuperMan extends Hero{
7      @Override
8      public String name() {
9          return " ";
10     }
11     public Hero hero() {
12         return new Hero();
13     }
14 }
15
16 public class SuperSuperMan extends SuperMan {
17     @Override
18     public String name() {
19         return " ";
20     }
21
22     @Override
23     public SuperMan hero() {
24         return new SuperMan();
25     }
26 }
```

java

What are variable length parameters?

Since Java 5, Java has supported the definition of variable-length parameters. Variable-length parameters allow you to pass in parameters of varying lengths when calling a method. For example, the following method can accept zero or more parameters.

```
1 public static void method1(String... args) {  
2     //.....  
3 }
```

java

In addition, a variable parameter can only be the last parameter of a function, but it may or may not be preceded by any other parameters.

```
1 public static void method2(String arg1, String... args) {  
2     //.....  
3 }
```

java

What should I do if I encounter method overloading? Will the method with fixed parameters or variable parameters be matched first?

The answer is that the method with fixed parameters will be matched first because the method with fixed parameters has a higher matching degree.

Let's demonstrate this through the following example.

```
1 /**  
2  *      JavaGuide      "      "  
3  *  
4  * @author Guide  
5  * @date 2021/12/13 16:52  
6  */  
7 public class VariableLengthArgument {  
8  
9     public static void printVariable(String... args) {  
10         for (String s : args) {  
11             System.out.println(s);  
12         }  
13     }  
14 }
```

java

```

15     public static void printVariable(String arg1, String arg2) {
16         System.out.println(arg1 + arg2);
17     }
18
19     public static void main(String[] args) {
20         printVariable("a", "b");
21         printVariable("a", "b", "c", "d");
22     }
23 }

```

Output:

```

1     ab
2     a
3     b
4     c
5     d

```

plain

class In addition, Java's variable parameters will actually be converted into an array after compilation, which can be seen from the files generated after compilation .

```

1     public class VariableLengthArgument {
2
3         public static void printVariable(String... args) {
4             String[] var1 = args;
5             int var2 = args.length;
6
7             for(int var3 = 0; var3 < var2; ++var3) {
8                 String s = var1[var3];
9                 System.out.println(s);
10            }
11        }
12        // .....
13    }
14

```

java

refer to

- What is the difference between JDK and JRE? <https://stackoverflow.com/questions/1906445/what-is-the-difference-between-jdk-and-jre>



- Oracle vs OpenJDK: <https://www.educba.com/oracle-vs-openjdk/>
- Differences between Oracle JDK and OpenJDK:
<https://stackoverflow.com/questions/22358071/differences-between-oracle-jdk-and-openjdk>
- Thoroughly understand Java's shift operators:
<https://juejin.cn/post/6844904025880526861>

JavaGuide官方公众号 (微信搜索JavaGuide)



- 1、公众号后台回复“PDF”获取原创PDF面试手册
- 2、公众号后台回复“学习路线”获取Java学习路线最新版
- 3、公众号后台回复“开源”获取优质Java开源项目合集
- 4、公众号后台回复“八股文”获取Java面试真题+面经

Recently Updated 2025/7/27 10:31

Contributors: Snailclimb , wangchao96 , SnailClimb , zeason , feng qijun , tecyang , Jin Yang , spike , Fenmul , dongzl , yguangyin , , catch , HanSai , yellowgg , hughyu , Kou Shuang , Kai , NaivePerdant , Yang Tian , Fy , shuang.kou , JerryQiang , yazhouasu , codeYu , das-friday-night , guide , Wenpeng , shellhub , LIU , VergeDX , TTL , TimorYang , Chong Chong Chong , Xiang Jaywhen , UniverseHua , sunguoliang , litho806 , 2293736867 , Tan Jiuding , kaka2634 , Snoopy , anaer , gongjixiaobai , Devhui , Zanzz , sam , Guide , ahh556 , Raxcl , daniubi , Amos Chu , cckrr , Horstson , Mr.Hope , dpod , Wang Xiaoqing , ZeroMarker , Dongcp , scwlkq , dongjunjie005 , will , qksuki , pcdd , qiliq , qq651901286 , xieliangza , Kisa-Dong , biea008 , AhogeK

Copyright © 2025 Guide

