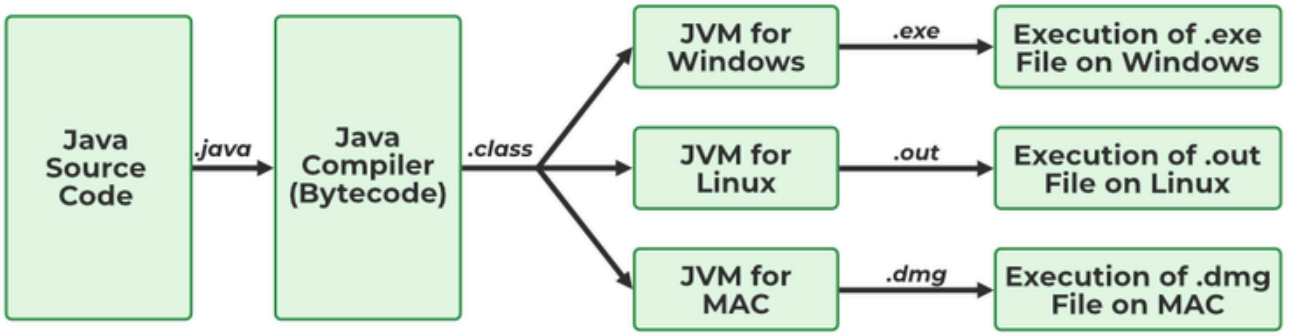
Java Basics & Program Flow

1. Is Java Platform Independent? If yes, how?

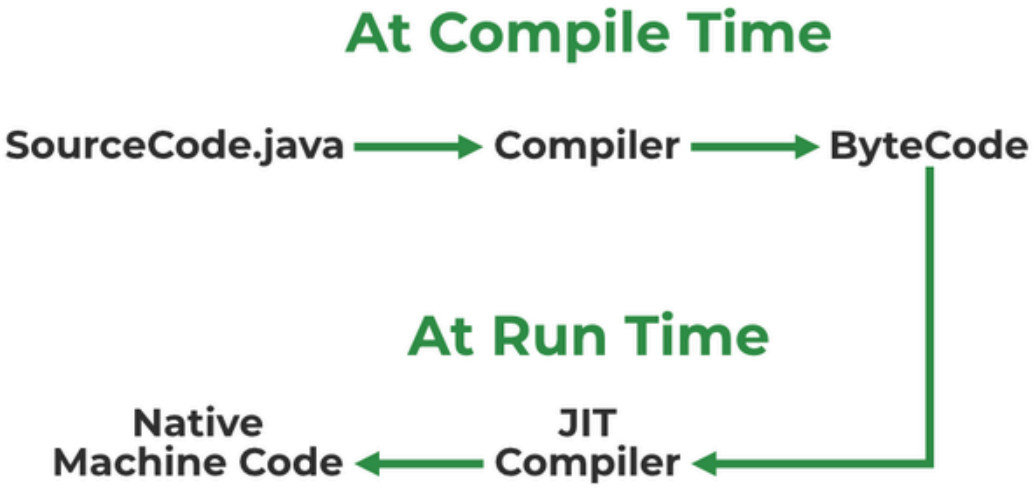
- Java source code is compiled by **javac** into **bytecode** stored in `.class` files.
- Bytecode is platform-Independent
- Each platform provides its own **JVM implementation**

2. What is a JVM?



- 执行Java byte code的引擎
- platform dependent，每个平台有自己的JVM
- **Loads classes (bytecode)** via **ClassLoader** -> 执行 bytecode
- Execution can be done by an **Interpreter** and optimized by **JIT** compilation to native machine code.

3. What is JIT?



JIT (Just-In-Time compiler) 是 JVM 里的即时编译器，用来提升运行时性能。

- Java 先变成 bytecode，运行时 JVM 会先解释执行，但如果某段代码经常被调用，JIT 会把它编译成 native machine code 并缓存，后面直接跑机器码，所以性能会更好

4. Difference between JVM, JRE and JDK.

JVM (Java Virtual Machine)

- 执行 **bytecode** 的虚拟机/引擎：class loading、bytecode verification、execution（Interpreter + JIT）
- **JVM 是平台相关的实现**（不同 OS）

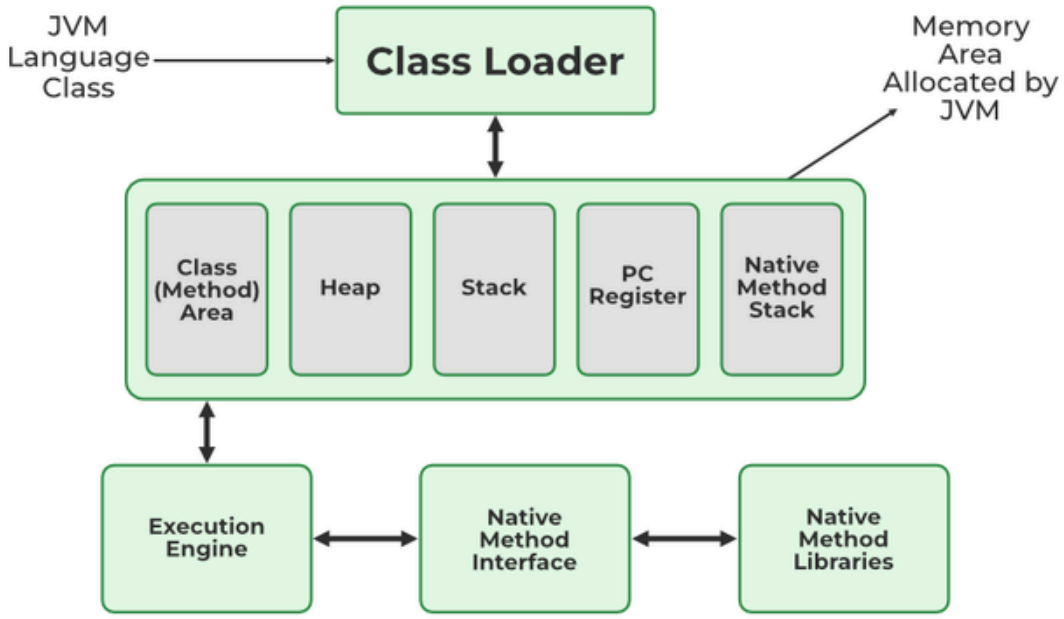
JRE (Java Runtime Environment)

- 用来“**运行** Java 程序”的运行环境。
- 本质组成：**JVM + Java standard libraries + runtime supporting files。**

JDK (Java Development Kit)

- 用来“**开发 + 运行**”Java 的开发工具包。
- 本质组成：**JRE + 开发工具**（如 `javac`，`jar`，`javadoc`，`jdb` 等）。
- 你要写代码、编译、打包、调试 → 用 JDK。

5. What are Memory storages available with JVM?



JVM 规范里核心运行时内存区（Runtime Data Areas）主要有 5 个：

Heap（堆，线程共享）

- 存放：几乎所有 **objects/arrays**。
- 特点：由 **Garbage Collection** 管理

Java Stack（虚拟机栈，线程私有）

每个线程一个栈，存放每次方法调用的 **stack frame**：

- 局部变量（local variables，基本类型值、对象引用 reference）
- 操作数栈（operand stack）
- 返回地址等

常见错误：递归太深 `StackOverflowError`；栈内存不足也可能 OOM。

PC Register（程序计数器，线程私有）

- 保存当前线程正在执行的字节码指令位置（下一条要执行的地址/偏移）。

Method Area（方法区，线程共享）

- 存放：类元信息（class metadata）、运行时常量池（runtime constant pool）、方法字节码、静态变量相关信息等。

Native Method Stack（本地方法栈，线程私有）

- 为 **JNI / native methods** 服务，存放 native 调用的栈帧。

注意：它不是“存 native 方法代码”，而是存 **native 调用过程的栈信息**。

6. what is class Loader?

- **ClassLoader** 是 JVM 的类加载系统，负责在运行时把 `.class` /模块里的 bytecode 读入并把它“定义”为 JVM 可用的 `Class` 。

7. Explain public static void main(String args[]) in Java.

`public static void main(String[] args)` 是 JVM 约定的程序入口（entry point）签名：

- **public**：JVM 需要从类外部调用它，因此必须可访问。
- **static**：无需创建对象实例就能调用入口方法。
- **void**：入口不返回值给 JVM（进程退出码通常用 `System.exit(code)` ）。
- **main**：方法名必须是 `main` （JVM 按约定查找）。
- **String[] args**：接收命令行参数。等价写法：`String... args` （varargs）。

8. What will happen if we don’t declare the main method as static?

- “main 不加 static 就成了实例方法。JVM 启动时没有对象，也不会自动 new 你的类，所以程序能编译但运行时会报 main 不是 static 或找不到入口方法。”

9. What are Packages in Java?

“Package 就是 Java 的命名空间和组织方式。它能避免类名冲突，并提供同包可见的访问控制（默认访问级别），同时让项目结构更清晰、便于复用和发布。”

12. How many types of packages are there in Java?

“通常分两类：Built-in packages（Java 标准库）和 User-defined packages（自己定义的包）。实际开发还会用很多第三方库的 packages。”

13. Why is Java not 100% Object-Oriented?

它有 primitive types，比如 int/boolean，它们不是对象、不继承 Object。需要对象行为时用 Integer 这些 wrapper classes，并且有 autoboxing/unboxing。”

Java Core Concepts

2. Differentiate between instance and local variables.

Instance Variable	Local Variable
Declared outside the method, directly invoked by the method.	Declared within the method.
Has a default value.	No default value
It can be used throughout the class.	The scope is limited to the method.

4. What is a Class Variable?

Class variable（类变量） 通常指 **static variable（静态变量）**：

- 用 `static` 修饰，**属于类本身**，不是某个对象。
- **同一个类在 JVM 里只有一份**（所有实例共享）。

6. Explain the difference between instance variable and a class variable.

- “实例变量是每个对象独有的状态；类变量是 static 的，属于类，所有对象共享。访问上实例变量靠对象，类变量推荐用类名。”

7. What is a static variable?

“static 变量属于类级别，所有对象共享一份，常用于共享状态或 `static final` 常量；如果会被多线程修改要注意同步。”

8. What is the difference between System.out, System.err, and System.in?

- “System.out 是标准输出 stdout，用来打印正常结果；System.err 是标准错误 stderr，用来输出错误信息，能和 out 分开重定向；System.in 是标准输入 stdin，用来读用户输入，常配合 Scanner。”

9. What is Java String Pool?

Java String Pool 是 **Heap** 中的一块特殊区域，用来存放 **string literals**（字符串字面量）。

- 使用 `"abc"` 这种方式创建的 String 会进入 **String Pool**
- JVM 创建 String 时会先检查 pool：
  - 如果 **已存在相同内容的 String** → 直接复用引用
  - 如果 **不存在** → 创建新对象并放入 pool
- 目的：**减少重复 String 对象，节省内存，提高性能**
- 注意：
  - `new String("abc")` **一定会创建新对象**（不复用 pool）
  - `String.intern()` 可以手动把字符串放入 pool

10. What do you understand by an IO stream?

“IO stream 是 Java 里进行输入输出的抽象，表示数据从哪里来、到哪里去。可以操作文件、网络、控制台等，并且可以通过包装流组合功能。”

11. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

InputStream / OutputStream

- 面向 **byte**
- 适合 **binary data**（图片、音频、视频、zip）
- 不处理字符编码

Reader / Writer

- 面向 **character**
- 适合 **text data**
- 自动处理 **character encoding（UTF-8 等）**

12. What are the super most classes for all the streams?

Java IO 四大顶级抽象类：

- **InputStream**：所有字节输入流父类
- **OutputStream**：所有字节输出流父类
- **Reader**：所有字符输入流父类
- **Writer**：所有字符输出流父类

13. What are the FileInputStream and FileOutputStream?

- **FileInputStream**
  - 从文件中 **按字节读取数据**
  - 常用于：图片、音频、二进制文件
- **FileOutputStream**
  - 向文件中 **按字节写数据**
  - 适合写 raw binary data

⚠️ 它们 **不做 buffering、不处理编码**，通常要配合 BufferedStream 使用。

14. What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

BufferedInputStream / BufferedOutputStream 提供 buffering

原理：

- 先把数据读入内存 buffer
- 再一次性处理，减少磁盘/网络 IO 次数

优点：**显著提高 IO 性能**

15. What are FilterStreams?

FilterStream 是一种 包装流（decorator）：

- 自身不直接读/写数据
- **包裹另一个 stream**，在读写过程中增加功能
- 常见子类：
  - `BufferedInputStream`
  - `DataInputStream`

- `CheckedInputStream`

NG 口语回答

“FilterStream 是 IO 里的装饰流，用来包裹已有 stream，在不改原 stream 的情况下增加功能，比如 buffering、数据格式支持等。”

16. What is an I/O filter?

- **I/O filter 指的就是 FilterStream 对象**
- 它从一个 stream 读取数据，对数据做处理，再传给下一个 stream
- 本质：**IO 层面的 decorator**

NG 口语回答

“I/O filter 就是 FilterStream，通过包装已有 stream，在数据流动过程中增加功能。”

17. How many ways you can take input from the console?

“常用有四种：命令行参数、Scanner、BufferedReader、Console。其中 Scanner 最常用，Console 适合安全输入。”

18. Difference in the use of print, println, and printf.

“print 不换行，println 会换行，printf 用格式化输出，是否换行取决于格式符。”

21. Explain the difference between >> and >>> operators.

- `>>`：**算术右移 (arithmetic shift right)**
  - 右移时左边补什么？补 **sign bit**（符号位）
  - 结果：保持正负号（负数右移仍可能是负数）
- `>>>`：**逻辑右移 (logical shift right)**
  - 右移时左边补什么？永远补 **0**
  - 结果：把数当成“无符号位模式”处理，负数右移会变成很大的正数

关键点：两者差异只在“左侧补位”：`>>` 补符号位，`>>>` 补 0。

26. When a byte datatype is used?

“byte 是 8 位有符号，-128 到 127。最常用于二进制数据读写，比如网络/文件；也可以在大量小范围数字时节省空间。”

27. What is the default value of byte datatype in Java?

“byte 作为成员变量默认是 0；但局部变量没有默认值，必须手动初始化。”

30. Why do we need wrapper classes?

“wrapper classes 把基本类型变成对象，主要是为了能放进 Collections/Generics，比如 List，同时提供 parse/valueOf 等工具方法，还支持 autoboxing/unboxing。”

31. What is autoboxing and unboxing?

Autoboxing：primitive → wrapper（如 `int` → `Integer`）自动发生

Unboxing：wrapper → primitive（如 `Integer` → `int`）自动发生

32. What is covariant return type?

“协变返回值就是重写时子类方法允许返回更具体的类型，比如父类返回 Number，子类可以返回 Integer。”

33. What is the transient keyword?

> **把 Java 对象转换成可以保存或传输的字节序列的过程**
> 反过来，把字节序列还原成 Java 对象，叫 **反序列化（Deserialization）**

“transient 表示这个字段不参与序列化，写文件/传输时会被跳过，反序列化后变成默认值。常用于敏感信息或可恢复的缓存字段。”

`transient` 用于 **serialization**：标记某个 field **不参与序列化。**

序列化时该字段不会写入流；反序列化回来后：

- 会变成该类型的 **default value**（如 0/false/null），除非你在反序列化后自己重新赋值。

35. What does it mean that Strings are immutable?

“String 不可变意味着内容不能改，拼接等操作会创建新对象。好处是线程安全、能用 String Pool 复用、也适合作为 HashMap key。”

36. What are the differences between String and StringBuffer?

“String 不可变；StringBuffer **可变而且 synchronized**，所以线程安全但性能较慢，适合多线程需要共享修改字符串的场景。”

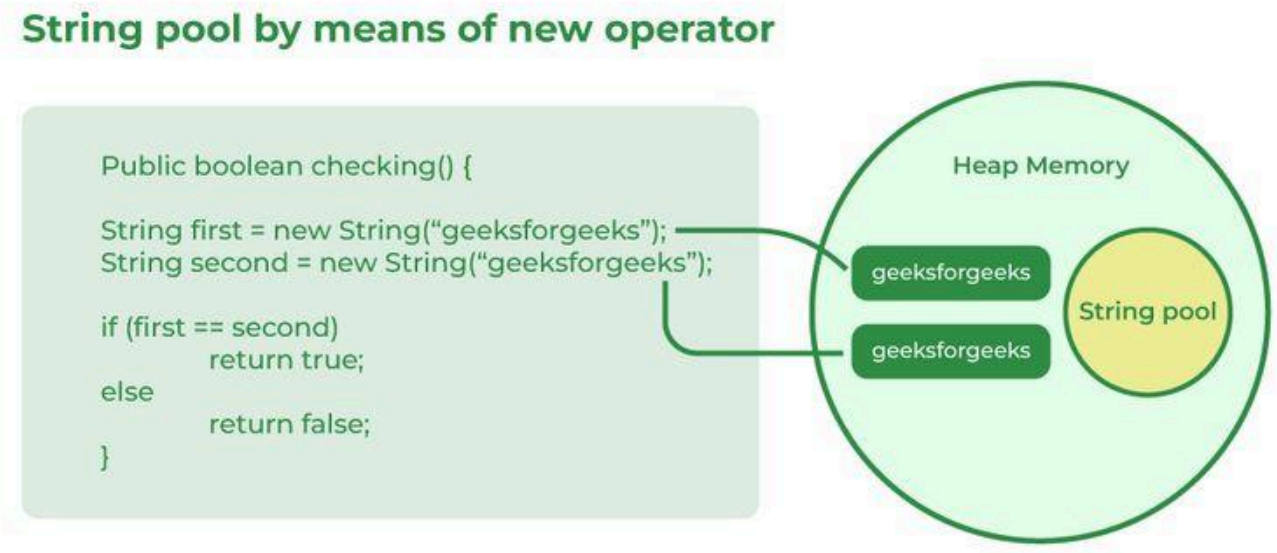
37. What are the differences between StringBuffer and StringBuilder?

“StringBuilder 更快但不线程安全；StringBuffer 线程安全但慢。单线程拼接通常用 StringBuilder。”

38. Why is StringBuffer called mutable?

“StringBuffer 是可变的，因为 append/insert 等是在原对象上改字符序列，减少创建新 String。”

39. How is the creation of a String using new() different from that of a literal?



"字面量会进入 String Pool，可能复用已有对象；new String 会强制在 heap 创建新对象，即使 pool 里已经有相同内容。"

42. On which memory arrays are created in Java?

In Java, arrays are always created in the **Heap** memory

47. What do you understand by the jagged array?

an array of arrays where each inner array can have a different length.

```
int[][] Arr = new int[][] {

    {1, 2, 8},

    {7, 5},

    {6, 7, 2, 6}

};
```

48. What are the advantages and disadvantages of an array?

"Array 的优点是随机访问快，O(1)，而且内存连续，cache 友好、开销小。缺点是大小固定，不能动态扩容，插入和删除需要移动元素是 O(n)，如果没用满还会浪费内存。"

Object-Oriented Programming (OOP)

1. What is an object-oriented paradigm and What are the main concepts of OOP in Java?

The main concepts of OOPs in Java are mentioned below:

👉 **“继多抽封”**

- **继** —— Inheritance（继承）
- **多** —— Polymorphism（多态）
- **抽** —— Abstraction（抽象）
- **封** —— Encapsulation（封装）

5. What are the different ways to create objects in Java?

NG 最稳的 4 类（够用且不踩坑）

1. `new` keyword **（最常用）**
   `Car c = new Car();`
2. **Reflection（反射）**
   - `Class.forName(...).getDeclaredConstructor().newInstance()`
     （✅ 推荐说这个，而不是老的 `Class.newInstance()`）
3. `clone()` **（克隆）**
   需要实现 `Cloneable` 并重写 `clone()`
4. `deserialization` **（反序列化）**

6. How is the ‘new’ operator different from the ‘newInstance()’ operator in Java?

`new`：编译期就写死类型，走正常构造流程
`newInstance()`：运行时通过 `reflection` 创建，类型可动态决定，并且异常更多/性能更慢

- This is part of the **Java Reflection API**. It allows you to create an object of a class even if you don't know the class name until the program is actually running.
- **Syntax:** `java Constructor<MyClass> constructor = MyClass.class.getConstructor(); MyClass obj = constructor.newInstance()`;

  Runtime Discovery: You can pass a string (like `"com.example.User"`) to load a class and create its instance at runtime.

7. What is the difference between static (class) method and instance method?

`static method` 属于 class；`instance method` 属于对象实例

8. What is this keyword in Java?

`this` 指向**当前对象实例**，用来访问当前对象的成员，或在 constructor 里调用另一个 constructor

```
Person() {
    this("Unknown", 0); // 必须是第一行
}

Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

9. What are the advantages and disadvantages of object cloning?

优点：快速复制对象结构；缺点：默认 shallow copy，且 API 设计麻烦、维护成本高

Pros（说 2 个够）

- `clone()` 创建**新对象**（不是只复制引用）
- 复制复杂对象时，能减少手写 copy 代码（类似 `Prototype pattern`）

Cons（背 3 个，面试够用）

- 需要实现 `Cloneable` + 重写 `clone()`，使用门槛高（`clone()` 还是 `protected`）
- 默认是 `shallow copy`，遇到引用字段容易踩坑
- 类结构一变（加字段/改嵌套对象），clone 逻辑容易漏，维护难

10. What is the difference between shallow cloning and deep cloning in Java?

`shallow copy`：只复制“外壳”，里面引用对象共享
`deep copy`：连内部嵌套对象也复制，互不影响

11. What are Access Specifiers and Types of Access Specifiers?

`public`：任何地方都能访问

`protected`：同包 + 子类可访问

`default`（不写就是它）：**同包可访问**

`private`：只有本类内部可访问

13. What is the constructor?

`constructor` 用来初始化对象；名字与类名相同；没有返回类型；在 `new` 时自动调用

14. How many types of constructors are used in Java.

`no-arg constructor`（无参）

`parameterized constructor`（有参）

`private constructor`（私有：用于 singleton / utility class 等）

15. What happens if we don't provide constructor in class?

如果你**一个 constructor 都没写**，Java 编译器会自动生成一个 `default constructor`（无参、空实现），并且会调用 `super()`。

如果你写了任何一个 constructor（比如有参的），编译器就**不会**再帮你生成无参的
→ 这时你如果还想要无参 constructor，必须自己写

16. What is the purpose of a default constructor?

`default constructor` 用来让类在**没有显式 constructor**时也能被 `new`，并把对象初始化为**默认状态**（并调用 `super()`）。

17. Where and how can you use a private constructor?

`private constructor` 用来**禁止外部 new**，控制对象创建方式，常见于 `Singleton`、`Utility class`、`Factory method`。

1. **Singleton pattern**

- 外部不能 `new`
- 通过 `getInstance()` 拿唯一实例

1. **Utility class**（只有静态方法的工具类）

- 比如 `Math` 风格
- 用 private constructor 防止被实例化

18. What are the differences between the constructors and methods?

`constructor` 负责创建/初始化对象；`method` 负责执行行为逻辑。

19. What do you mean by data encapsulation?

`encapsulation` 是把数据(fields)和操作数据的方法(methods)封装在 class 里，并通过 `access modifiers` 控制访问，实现 `data hiding`。

20. What are the advantages of Encapsulation in Java?

`encapsulation` 的核心价值是 `data hiding` + 保持 `invariants`，让代码更安全、可维护、可扩展。

23. What is inheritance in Java?

`inheritance` 让 `subclass` 通过 `extends` 复用 `superclass` 的 fields/methods，并支持 `polymorphism`（多态）。

24. What are the different types of inheritance in Java

- Single Inheritance：**一个孩子只有一个父母。**
- Multilevel Inheritance：**爷爷 → 父亲 → 儿子，一代一代传。**
- Hierarchical Inheritance：👉 **一个父类，多个孩子。**多个子类 **继承同一个父类。**
- Multiple Inheritance：👉 **一个孩子有多个父母。**
  - 类对类：❌不支持
  - interface：✅支持（`implements` 多个）

25. What is multiple inheritance? Is it supported by Java?

`multiple inheritance` 是一个 class 继承多个 parent classes；Java **不支持 classes 的 multiple inheritance**，但支持 `multiple interfaces`。

26. Is there any limitation to using Inheritance?

- Tight coupling：父类改动会影响子类
- Fragile base class problem：父类新增/修改方法可能改变子类行为

27. What is the 'IS-A ' relationship in OOPs Java?

`IS-A` 表示继承关系：`Dog IS-A Animal`，通常用 `extends` 或 `implements` 表达，并支持 `polymorphism`。

28. What is HAS-A relationship in OOP (Aggregation/Composition)?

`HAS-A` 表示组合关系：一个类把另一个类当成员变量；通常更灵活，优先于 inheritance。

Aggregation vs Composition（背这对比就够）

- `Aggregation`：弱拥有（对象可独立存在）
  例：`Team` has `Player`，player 可以离开 team 还存在
- `Composition`：强拥有（生命周期绑定）
  例：`Car` has `Engine`，car 没了 engine 就没意义/一起消亡（概念上）

> 注意：Java 语法层面并没有关键字区分这俩，主要是**设计语义 + 生命周期**。

29. What do you mean by aggregation?

`aggregation` 是一种 `HAS-A` 的 association，表示"整体-部分"关系，但部分对象能独立存在，属于弱拥有。

30. What is an association?

`association` 是两个 class 之间通过对象建立的关系，是更泛化的概念；`aggregation` / `composition` 都是 association 的更具体形式。

31. What is the composition of Java and State the difference between Composition and Aggregation?

`composition` 是一种强 `HAS-A` 关系：**parent 拥有 child 的生命周期**，parent 没了 child 也就不该存在；`aggregation` 是弱 `HAS-A`：child 可以独立存在。

Composition（强拥有）

- child **不能独立存在**（conceptually tied）
- parent 负责创建/销毁 child（lifecycle bound）
- 典型：`Car` has `Engine`（强组合语义）

Aggregation（弱拥有）

- child **可以独立存在**
- parent 只是“引用”或“使用”child
- 典型：`Team` has `Player`（player 离开 team 仍存在）

32. What is Polymorphism?

同一个 `interface` / `reference type` 在运行时指向不同具体对象时，调用同名 `method` 表现不同的能力（核心靠 `method overriding` + `dynamic dispatch`）。

你用父类类型 `Animal` 去接 `Dog` 或 `Cat`，调用 `sound()`，真正执行哪个版本由**对象真实类型**决定。

33. How many types of Polymorphism.

Java 常说两类：`compile-time polymorphism`（`overloading`）和 `runtime polymorphism`（`overriding` / `dynamic dispatch`）。

32. What is Compile-time (Static) Polymorphism?

`compile-time polymorphism` 通常指 `method overloading`：同名方法但参数列表不同，编译器在编译期根据参数类型决定调用哪个。

33. What is Runtime (Dynamic) Polymorphism?

`runtime polymorphism` 也叫 `dynamic method dispatch`：当子类 `override` 父类方法时，实际执行哪个实现由运行时对象类型决定。

34. What is method overriding and method overloading?

Overloading（同名不同参）

- 同一个 class 里（或父子类也可以，但看参数区分）
- 只看参数列表不同
- 编译期绑定（`compile-time`）

Overriding（重写）

- 必须是父子类关系（`extends`）
- `method signature` 必须相同（参数相同）
- `return type` 可以是 `covariant return type`（子类返回更具体类型）
- 运行期绑定（`runtime`）

35. Can we override the static method ?

不能。`static method` 属于 class，不参与 `polymorphism`；子类只能做 `method hiding`，不是 `overriding`。

36. Can we change the scope of the overridden method in the subclass?

可以，但只能变得**更宽松**（more accessible），不能更严格。

父类 `public` → 子类只能 `public`

父类 `protected` → 子类 `protected` 或 `public`

父类 `default`（package-private）→ 子类 `default` / `protected` / `public`

父类 `private` → **不能** override（对子类不可见）

37. What is Abstraction?

For example, **when you use a coffee maker, the internal mechanisms—such as water heating and the brewing process—are abstracted from you**. You don't need to understand how the machine achieves these functions, only that the task is completed when you follow the prescribed steps.

`abstraction` 是只**暴露对外需要的行为（what）**，隐藏**内部实现细节（how）**，让使用者只关心接口而不是实现。

- 使用者只知道 “**这个对象能做什么**”
- 不需要知道 “**它内部是怎么实现的**”
- 通过 `abstract class` 或 `interface` 实现
- 好处是：**实现可替换，使用方不受影响**

38. How many ways to achieve abstraction in Java.

Java 主要通过 **`abstract class` 和 `interface`** 实现 abstraction，但 **interface 并不是 100% abstraction**（Java 8 之后）。

正确、稳妥的说法（背）

- `abstract class`： **partial abstraction**

- `interface`：**high-level abstraction**（但不再是 100%）

为什么 interface 不是 100% abstraction?

因为 Java 8+：

- `default methods` 有实现
- `static methods` 也有实现

39. What is Abstract class?

`abstract class` 是不能被实例化的 class，用来作为 base class，既可以包含 `abstract methods`，也可以包含已实现的方法和成员变量。

核心规则（NG 必背）

- 不能 `new`（不能直接创建对象）
- **可以没有** abstract method
- 但：**只要有** abstract method，class **必须是** abstract
- 子类必须 `override` 所有 abstract methods（否则子类也得是 abstract）

```
abstract class Fruits {
    abstract void run();
}

class Apple extends Fruits {
    void run() {
        System.out.println("Abstract class example");
    }
}
```

Abstraction

- 关注：**what**
- 隐藏：实现细节
- 手段：`abstract class` / `interface`

40. What is an Interface?

`interface` 是一种 **contract**，只定义类**必须具备的行为（what）**，而不关心具体实现（how），实现它的 class 必须实现所有 required methods。

- `interface` 定义 **行为规范**
- class 通过 `implements` 来承诺遵守这个规范

41. Give some features of the Interface.

`interface` 提供高层 abstraction，支持 `multiple inheritance`（通过接口），并促进 `loose coupling`。

42. What is a marker interface?

`marker interface` 是一个 **没有任何 methods 或 fields 的 interface**，用于向 JVM 或 framework **标记某种能力或属性。**

常见例子（必背）

- `Serializable`
- `Cloneable`
- `RandomAccess`

43. What are the differences between abstract class and interface?

`abstract class`： **IS-A + code reuse**

`interface`： **CAN-DO + contract**

对比点	Abstract Class	Interface
关键字	`abstract`	`interface`
继承/实现	`extends`（只能一个）	`implements`（可多个）
Multiple inheritance	❌（class）	✅（interface）
Methods	abstract + concrete	abstract（默认）+ `default` / `static`（Java 8+）
Fields	任意 access	默认 `public static final`
Access modifier	可 `private/protected`	methods 默认 `public`
Constructors	✅ 有	❌ 没有
State (fields)	✅ 可有实例变量	❌ 不能有实例变量
使用场景	closely related classes	unrelated classes

Collections & Generics

2. What is the difference between Collection and Collections?

- Collection 是 java.util 里的接口，表示一组元素，是 List/Set/Queue 的父接口，定义了 add、remove、size 等基本行为。
- Collections 是 java.util 里的工具类（utility class），提供一堆 static 方法来操作 Collection，比如 sort、reverse、shuffle、synchronizedList、unmodifiableList 等。

6. How can you synchronize an ArrayList in Java?

(1) 用 Collections.synchronizedList 把它包装成同步 List：List sync = Collections.synchronizedList(list)。

📌 发生了什么?

- 原来的 `list`（比如 `ArrayList`）**不是线程安全的**
- `synchronizedList` 给它**套了一层壳**
- 每个方法都变成：

```
synchronized (mutex) {
    原来的方法();
}
```

所以：

```
sync.add(x);    // 安全
sync.get(i);    // 安全
sync.remove(i);// 安全
```

注意遍历时也要手动 synchronized(sync) 来保证迭代期间不被并发修改。

(2) 用 java.util.concurrent 的 CopyOnWriteArrayList，它在写入时拷贝数组，读操作无锁，适合读多写少场景。

- 读的时候不用加锁（很快），写的时候会复制一份新数组（比较慢），所以非常适合「读很多、写很少」的场景。

7. Why do we need a synchronized ArrayList when we have Vectors (which are synchronized) in Java?

> “带锁” = 同一时间，只允许一个线程进来用这个东西，
> 其他线程必须在门口排队等。

- Vector = “**永远带锁**” → 不管用不用，都慢
- ArrayList + synchronized = “**需要的时候才加锁**” → 更灵活、通常更快

8. Why can’t we create a generic array?

不能直接创建泛型数组（比如 new T[]），因为数组在运行时会保存并检查元素类型（有 ArrayStoreException 的运行时检查），但泛型在编译后会发生 type erasure，运行时看不到 T 的真实类型。这样 JVM 无法对 T[] 做正确的类型检查，会产生类型不安全，因此 Java 禁止创建泛型数组。

10. How do you convert an ArrayList to an Array and an Array to an ArrayList in Java?

- Array 转 List：用 Arrays.asList(array)，它返回一个“固定长度”的 List 视图，不能 add/remove；如果想要真正可变的 ArrayList，可以 new ArrayList<>(Arrays.asList(array))。
- List 转 Array：用 list.toArray(new T[0])（或 new T[list.size()]），得到类型正确的数组；如果用无参 toArray() 会返回 Object[]。

12. What is a Vector in Java?

Vector 就像老版本的 ArrayList：

- 动态数组
- **自带同步**（线程安全但慢）
- 现在更多推荐：ArrayList + 需要时同步，或用并发集合

13. How to make Java ArrayList Read-Only?

- 给它套个“保护壳”：Collections.unmodifiableList(list)
- 你再 add/remove 就会抛 UnsupportedOperationException

15. What is the Stack class in Java and what are the various methods provided by it?

- **Stack 是 Java 里的 LIFO（后进先出）栈结构，继承自 Vector。常用方法：push 入栈，pop 出栈并返回，peek 查看栈顶不移除，empty 判断是否为空，search 返回元素距栈顶的 1-based 位置（找不到返回 -1）。**

16. What is Set Interface?

Set 是 java.util 里的接口，表示“不重复元素的集合”：同一个元素最多出现一次。它通常不保证下标访问；是否保持顺序/是否排序取决于实现（HashSet 无序，LinkedHashSet 保插入顺序，TreeSet 排序）。大多数实现允许最多一个 null，但 TreeSet 一般不允许 null（因为要比较/排序）。

17. What are the implementation classes of the set interface?

常见 Set 实现：

- **HashSet（无序，基于哈希，允许一个 null）**
- **LinkedHashSet（保持插入顺序）、**
- **TreeSet（有序/排序，基于红黑树，通常不允许 null）、**
- **EnumSet（专门给 enum 用，性能高，不允许 null）。**

18. What is the HashSet class in Java and how does it store elements?

HashSet 实现了 Set：去重、通常不保证顺序。内部实际用 HashMap 存储元素：把元素当作 key，value 放一个固定的哑对象。插入时先用 hashCode 定位桶，再用 equals 判断是否已存在，从而保证唯一。发生哈希冲突时会用链表/（Java 8+ 可能树化为平衡树）来处理。

19. What is LinkedHashSet in Java Collections Framework?

LinkedHashSet 是 HashSet 的有序版本：在哈希表基础上维护一条双向链表来记录插入顺序，因此迭代顺序与插入顺序一致；去重规则仍然依赖 hashCode + equals。

22. What is a priority queue in Java?

PriorityQueue 是基于堆（heap）的队列实现，出队顺序由优先级决定：默认是“最小堆”，peek/poll 拿到当前最小元素；也可以用 Comparator 定义规则变成最大堆或自定义优先级。它不保证遍历输出是有序的，只保证队首是最高优先级。

23. What is BlockingQueue?

> **`BlockingQueue` 是一个线程安全的队列：**
> **队列空时，取数据的人会“等着”；**
> **队列满时，放数据的人也会“等着”。**

👉 **等着 = 自动阻塞（blocking）**

24. What is a Map interface in Java?

Map 是 java.util 里的接口（注意：它不属于 Collection 的子接口），用于保存 key-value 映射。key 必须唯一，value 可以重复。常用方法有 put/get/containsKey/containsValue/remove。常见实现包括 HashMap、LinkedHashMap、TreeMap、ConcurrentHashMap。

25. What is TreeMap? Explain internal working.

TreeMap 是有序的 Map，按 key 的自然顺序或 Comparator 排序。内部用红黑树（自平衡二叉搜索树）存储，因此 get/put/remove 的时间复杂度通常是 O(log n)。

26. What is EnumSet?

10岁理解

EnumSet 是“**给 enum 专用的 Set**”，特别快。

满分背诵版

EnumSet 是针对 enum 类型的高性能 Set 实现：只能存同一种枚举类型的值，内部通常用位向量/bitset 表示，所以非常快且省内存。它不允许 null，且是非同步的（非线程安全）。

28. What is ConcurrentHashMap in Java?

10岁理解

它是“**多人同时用也不会乱的 HashMap**”，而且不会整张表都锁死。

满分背诵版

ConcurrentHashMap 是线程安全的 Map 实现，支持高并发读写。相比 Hashtable 或给 HashMap 整体加锁，它不会锁住整个 map，而是用更细粒度的并发控制（现代实现主要用 CAS + synchronized 组合）来提高吞吐量，因此非常适合多线程场景。并且它不允许 null key 或 null value。

29. Can you use any class as a Map key?

理论上任何类都能作为 Map 的 key，但必须正确实现 equals() 和 hashCode()（保证相等对象 hashCode 相同），并且 key 最好是不可变（immutable），否则放进去后如果字段变了，会导致定位桶变化，出现“拿不出来”的问题。

30. What is a cursor in Java Collections

10岁理解

cursor 就是“**用来挨个走过集合元素的指针/遍历器**”。

满分背诵版

Cursor 是遍历集合元素的对象。Java 常见三种：

- Enumeration：老版本（Vector/Hashtable），只能读（hasMoreElements/nextElement）。
- Iterator：通用，能遍历并支持 remove。
- ListIterator：专用于 List，支持双向遍历（previous/next），还能 add/set。**

31. What is an Iterator?

Iterator 是 java.util 的遍历接口，用来按顺序遍历任何 Collection。通过 collection.iterator() 拿到迭代器，然后用 hasNext()/next() 访问元素，并且可以用 iterator.remove() 在遍历过程中安全地删除“刚刚返回的那个元素”。它是 Enumeration 的现代替代。

33. What is the difference between Array and Collection in Java?

Array 是“固定盒子”；Collection 是“一整套更强的盒子系统（List/Set/Queue）”，通常可以动态变大变小，还自带很多高级操作。

35. Differentiate between ArrayList and Vector in Java.

Vector 像“老式的线程安全 ArrayList”，默认每次操作都加锁，所以慢；ArrayList 默认不加锁，所以快。

36. What is the difference between Iterator and ListIterator?

Iterator 适用于所有 Collection，单向遍历（hasNext/next），支持 remove。ListIterator 只用于 List，支持双向遍历（previous/hasPrevious），并支持 add/set、以及 nextIndex/previousIndex 获取位置。

37. Differentiate between HashMap and HashTable.

10岁理解

Hashtable 是“老的、默认加锁的 map”，不允许 null；HashMap 是“新的、更常用的 map”，不默认加锁，允许 null。

满分背诵版

HashMap 非同步，允许一个 null key 和多个 null value，性能通常更好；Hashtable 同步（synchronized），不允许 null key/value，属于旧类。并发场景一般推荐 ConcurrentHashMap，而不是 Hashtable。

38. What is the difference between Iterator and Enumeration?

10岁理解

Enumeration 是老版本的“只读遍历器”；Iterator 是新版本的“可删除遍历器”，还带 fail-fast 检测。

满分背诵版

Enumeration 是 legacy 游标（Vector/Hashtable），只有 hasMoreElements/nextElement，不能 remove；Iterator 是现代游标，适用于所有集合，支持 remove，并且多数实现是 fail-fast（并发修改会抛 ConcurrentModificationException）。

39. What is the difference between Comparable and Comparator?

10岁理解

- Comparable：物品“自己会比大小”（类自己定义天然排序）。
- Comparator：外部拿一把“尺子/规则”，想按什么排序就给什么规则（可多种）。

满分背诵版

Comparable（java.lang）由对象自身实现 compareTo，用于定义“自然顺序”（一种默认排序）。Comparator（java.util）是外部比较器，实现 compare(o1,o2)，用于自定义/多种排序规则，不需要修改原类。TreeSet/TreeMap 或 Collections.sort 都可以用 Comparator 来指定排序。

41. Explain the FailFast iterator and FailSafe iterator along with examples for each.

10岁理解

- **Fail-Fast**：你在遍历时偷偷改集合，它会立刻“炸掉”提醒你（抛异常）。
- **Fail-Safe / Weakly Consistent**：遍历时允许别人改，它不会炸，但你看到的可能不是最新全量。

满分背诵版（建议背这一段）

Fail-Fast 迭代器会在检测到遍历期间集合被“结构性修改”（非 iterator.remove）时抛 ConcurrentModificationException，例如 ArrayList、HashMap 的 iterator。
并发容器（如 ConcurrentHashMap、CopyOnWriteArrayList）的迭代器通常是弱一致性（常被口头称为 fail-safe）：遍历时允许并发修改，不抛 CME，但不保证实时看到所有修改；CopyOnWriteArrayList 迭代器更像快照（snapshot），遍历的是创建迭代器时的副本。

Advanced Topics

1. What is Exception Handling & How many types of exceptions can occur in a Java program?

Exception Handling = **程序出问题时，不让它直接崩掉，而是用 `try/catch/finally/throws` 把错误“接住、处理、或往上交”。**

关键点（背这个）

- Java 的异常顶层都是 `Throwable`
- 两大分支：`Exception`（程序可以处理） vs `Error`（JVM 严重问题）

- `Exception` 里常说的两类：**Checked** & **Unchecked(RuntimeException)**
- 还有一种分类：**Built-in**（Java自带） vs **User-defined**（你自定义）

2. Difference between an Error and an Exception.

Error：系统级/ JVM 级严重问题，通常不可恢复；Exception：业务/程序级问题，通常可处理或可恢复。

3. Explain the hierarchy of Java Exception classes.

背诵版结构

- `Object`
  - `Throwable`
    - `Error`
    - `Exception`
      - `RuntimeException`（unchecked）
      - 其它 `Exception`（通常是 checked，比如 `IOException`）

4. Explain Runtime Exceptions.

运行时异常（RuntimeException）= 不强制编译期处理，通常代表代码 bug。

- Unchecked：编译器不逼你 `try/catch` 或 `throws`
- 常见：`NullPointerException`，`ArrayIndexOutOfBoundsException`，`IllegalArgumentException`

5. What is NullPointerException?

你对一个“空对象引用（null）”做了任何需要对象的操作，就会 NPE。

6. When is the ArrayStoreException thrown?

你把“错误类型”的对象塞进“运行时类型不匹配”的数组里

7. What is the difference between Checked Exception and Unchecked Exception?

Checked

- 编译期检查：必须 `try/catch` 或 `throws`
- 多是外部环境导致：文件、网络、数据库
- 例子：`IOException`，`SQLException`，`InterruptedException`

Unchecked (RuntimeException)

- 编译期不检查
- 多是代码逻辑问题：空指针、越界、参数非法
- 例子：`NullPointerException`，`IndexOutOfBoundsException`，`IllegalArgumentException`

8. What is the base class for Error and Exception?

共同父类是 `java.lang.Throwable`。

9. Is it necessary that each try block must be followed by a catch block?

不一定。`try` 后面可以是 `catch`，也可以是 `finally`，也可以 `catch + finally`。但不能只有 try。

10. What is exception propagation?

一个方法没抓住异常，就把异常“往上扔”给调用它的方法 (caller)，一层层往上，直到被 catch 或者程序崩。

11. What will happen if you put System.exit(0) on the try or catch block? Will finally block execute?

`System.exit(0)` 会**直接让 JVM 退出**，所以一般情况下 **finally 不会执行**。

12. What do you understand by Object Cloning and how do you achieve it in Java?

Cloning = 复制对象。Java 传统做法是实现 `Cloneable` + 重写 `clone()`，但默认是**浅拷贝** (shallow copy)：只复制字段值；字段里如果是引用对象，引用地址还是同一个。

- `Cloneable` 是 **marker interface**（空接口，表示“允许 clone”）

13. How do exceptions affect the program if it doesn't handle them?

异常如果没被 catch，会**向上抛（propagate）**，最终导致**当前线程终止**。

14. What purpose do the keywords final, finally, and finalize fulfill?

- `final`：“**不能改/不能被重写/不能被继承**”
- `finally`：异常处理里“**不管有没有异常都执行（通常）**”
- `finalize()`：以前 GC 前可能调用的清理方法，但**不可靠且已不推荐/已废弃方向**（工程上用 `try-with-resources` / `AutoCloseable` / `Cleaner`）

15. What is the difference between this() and super() in Java?

秒懂版

- `this()`：调用**本类**的另一个构造器
- `super()`：调用**父类**的构造器

背诵要点

- 都必须是构造器里的**第一行**
- 一个构造器里 `this()` 和 `super()` **只能选一个**（因为都要第一行）
    - 构造器在开始执行前，必须“先确定到底从哪一个构造器链开始”，而这个入口只能有一个。
- 如果你不写 `super()`，编译器会默认插入无参 `super()`

16. What is multitasking?

> Concurrency（并发）
>
> - 重点：**任务交替推进**
> - 可能只有 **1 个 CPU 核**，也能并发（靠切换）
>
> 例子：你一边下载文件一边响应 UI 点击——其实可能是在轮流切换。
>
> Parallelism（并行）
>
> - 重点：**真的同时运行**
> - 需要 **多核** CPU 或者多机器
>
> 例子：你有 8 核，8 个线程真同时在 8 核上跑。
>
> ✅ Java 里你写多线程，**不一定并行**，但一定是并发结构。

意思是：**同一段时间内，看起来在做很多事。**
在计算机里通常指：

- **一个进程(process)** 里跑多个任务（比如多个线程）
- 或 **操作系统** 让多个进程轮流跑（时间片切换）

> 你在 Java 里说 multitask，面试官一般默认你说的是 **concurrency（并发）**：把很多任务“组织起来同时推进”。

17. What do you mean by a Multithreaded program?

- ✅ 关系：
  multithread 是实现 multitask 的一种主要手段（尤其在 Java 里）。
- **multithreaded program = 一个进程里有多个 threads 并发 concurrent执行**，不是一条路走到底。
  - 你要是multi core CPU，也可以是parallel

18. What are the advantages of multithreading?

- Responsiveness：UI/服务不被阻塞
- Throughput：多个请求/任务并行处理
- Resource sharing：共享内存通信更快
- Scalability：多核上可以并行
- Trade-off：race condition、deadlock、context switch 开销

19. What are the two ways in which Thread can be created?

满分口语（English）

> There are two classic ways: extend `Thread` and override `run()`, or implement `Runnable` and pass it to a `Thread`.
> In modern Java, we usually prefer `Runnable` (or `Callable`) with an `ExecutorService` so we can use thread

> pools and manage resources better.

20. What is a thread?

一个process里有很多thread，thread是最小执行单位。

Thread = 程序里最小的执行单元。每个 thread 有自己的 **stack / program counter / local variables**，但同一个进程内线程共享 **heap（对象内存）**。

- 每线程独有：stack、PC、locals
- 线程共享：heap、文件句柄等进程资源
- 风险：共享数据 → 需要同步（locks/volatile/atomic）

21. Differentiate between process and thread?

- **Process（进程）**：一个正在运行的程序实例，有**独立内存空间**
- **Thread（线程）**：进程里的“执行路线”，多个线程**共享同一块内存（heap）**
    - 隔离：Process 隔离强；Thread 共享内存

      开销：Process 创建/切换更贵；Thread 更轻量

      通信：Process 需要 IPC（慢）；Thread 共享内存（快，但要同步）

22. Describe the life cycle of the thread?

线程从“出生”到“结束”大概会经历：**New → Runnable → (Blocked/Waiting/Timed Waiting) → Terminated**

背诵要点

- **NEW**：刚 `new Thread()`，还没 `start()`
- **RUNNABLE**：可运行（可能正在跑，也可能在就绪队列等 CPU）
- **BLOCKED**：等锁（等进入 synchronized）
- **WAITING**：一直等别人叫醒（`wait()` / `join()` 无超时）
- **TIMED_WAITING**：等一段时间（`sleep()` / `wait(timeout)` / `join(timeout)`）
- **TERMINATED**：跑完 run() 或异常结束

23. Explain suspend() method under the Thread class.

`suspend()` 会**强制暂停线程**，但它可能在你拿着锁的时候把你停住，导致别的线程永远拿不到锁 → **死锁**，所以被废弃。

24. Explain the main thread under Thread class execution.

程序一启动 JVM 就创建一个 **main thread** 来执行 `public static void main(...)`。你创建的其他线程通常都是从它“派生”出来的。

- main 结束 ≠ JVM 一定退出：只要还有非 daemon 线程在跑，JVM 还会活着

25. What is a daemon thread?

A daemon thread is a background service thread. 当程序里只剩 daemon 线程时，JVM 会直接退出，不会等它们做完。

背诵要点

- 用途：后台清理/监控/心跳（例如 GC 相关线程、某些后台调度）

26. What are the ways in which a thread can enter the waiting state?

满分口语（English）

> A thread can enter waiting states in several ways:
>
> - `sleep()` puts it into `TIMED_WAITING`.
> - `Object.wait()` releases the monitor and goes to `WAITING` (or `TIMED_WAITING` with a timeout) until `notify/notifyAll`.
> - `join()` makes the current thread wait until another thread finishes.
>
> Some blocking operations like I/O can also block the thread depending on the underlying implementation.

27. How does multi-threading take place on a computer with a single CPU?

单核没有真正同时跑多个线程，只是 CPU **快速轮流切换（time slicing）**，让你感觉“同时”。

背诵要点

- Concurrency ≠ Parallelism
- OS scheduler 负责切换线程

28. What are the different types of Thread Priorities in Java? And what is the default priority of a thread assigned by JVM?

线程优先级是 1–10：

- `MIN_PRIORITY = 1`
- `NORM_PRIORITY = 5`（默认）
- `MAX_PRIORITY = 10`

29. Why Garbage Collection is necessary in Java?

因为 Java 的对象都在 heap 上创建，如果不用的对象不回收，就会**内存越堆越大**，最终 OutOfMemoryError。GC 就是自动把“没人再用”的对象清掉。

30. What is the drawback of Garbage Collection?

秒懂版

GC 的最大缺点：**会产生暂停（pause）和不可预测性**，并且会带来额外 CPU/内存开销。

背诵要点

- **Pause / Latency**：Stop-the-world 暂停（尤其对低延迟系统敏感）
- **Non-deterministic**：什么时候 GC、暂停多久不完全可控
- **Overhead**：GC 本身占 CPU；可能需要更多内存（为了吞吐/分代等）

31. Explain the difference between a minor, major, and full garbage collection.

秒懂版（中文）

把 Java 堆（heap）想成两块区域：

- **Young Gen（新生代）**：新对象大多先来这
- **Old Gen（老年代）**：活得久的对象会被“升职”到这

对应 GC：

- **Minor GC**：只清 **Young Gen**（主要是 Eden + Survivor）
- **Major GC**：清 **Old Gen**（老年代回收）
- **Full GC**：**整个堆**都清（Young + Old，通常还可能带上 Metaspace 的回收/整理）

32. How will you identify major and minor garbage collections in Java?

开 GC 日志后，看关键词就行：

- 看到 **Young / Minor / Pause Young / Evacuation** → 基本是 **Minor GC**
- 看到 **Old / Major / Mixed**（G1 里 mixed 会同时动 young+部分 old）→ **老年代相关**
- 看到 **Full GC / Pause Full** → **Full GC**

33. What is a memory leak, and how does it affect garbage collection?

memory leak（内存泄漏）在 Java 里不是“忘记 free”，而是：

> 对象明明不用了，但还被某个引用“拴着”（strong reference），GC 认为它还活着 → 回收不了。

结果：

- 堆越用越大 → GC 越来越频繁
- 暂停更长、吞吐下降
- 最后可能 **OutOfMemoryError**

36. What is JDBC?

JDBC（Java Database Connectivity）就是 Java 访问关系型数据库（MySQL/PostgreSQL/Oracle…）的标准接口：
Java 程序通过 JDBC API 发 SQL、拿结果。

> **JDBC 属于 Java（JDK 的一部分）**
> **JDBC Driver 由数据库厂商提供，但运行在 Java 里，是 Java 与数据库之间的桥梁**

背诵要点

- JDBC = 标准 API（接口/规范）

- 具体连哪种数据库靠 **Driver（驱动）**

37. What is JDBC Driver?

Driver 是“翻译官”：把 JDBC 的统一调用翻译成某个数据库懂的协议。

四类（面试记 Type 就行）：

- **Type 1**：JDBC-ODBC Bridge（已淘汰，现代基本不用）
- **Type 2**：Native-API（依赖本地库）
- **Type 3**：Network Protocol（中间层服务器转发）
- **Type 4**：Thin / Pure Java（直接用数据库协议，最常用）

> A JDBC driver is the component that implements the JDBC API for a specific database.
>
> There are four historical types, and in modern systems we mostly use **Type 4 (thin, pure Java)** drivers because they communicate directly with the database using its native protocol and don’t require native libraries.

38. What are the steps to connect to the database in Java?

更标准：

1. 获取 `Connection`（通过 `DriverManager` 或更推荐的 `DataSource`）
2. 创建 `PreparedStatement`（推荐，防 SQL 注入）
3. `executeQuery()` / `executeUpdate()`
4. 读 `ResultSet`
5. 关闭资源（推荐 `try-with-resources`）

39. What are the JDBC API components?

面试最常答这一套就够：

- `Driver` / `DriverManager` 或 `DataSource`
- `Connection`
- `Statement` / `PreparedStatement` / `CallableStatement`
- `ResultSet`
- `SQLException`

40. What is JDBC Connection interface?

`Connection` 表示你和数据库之间的一条“会话通道”。
它负责：

- 开启/关闭连接
- 事务：`commit()` / `rollback()`、`setAutoCommit(false)`
- 隔离级别：`setTransactionIsolation(...)`

- 创建 statement：`prepareStatement(...)`

41. What does the JDBC ResultSet interface?

`ResultSet` 就是 SQL 查询返回的“表格结果”，带一个**游标 cursor**。
你用 `next()` 一行一行往下走，然后用 `getInt/getString` 取列值。

42. What is the JDBC Rowset?

RowSet 可以理解为 “**更高级的 ResultSet**”：
它也是表格数据，但通常更适合做：

- **可序列化 / 可断开连接（CachedRowSet）**
- 更方便与 UI/组件集成

常见类型（知道 1–2 个就够）：

- `JdbcRowSet`（连接型）
- `CachedRowSet`（断开型，常被提）

43. What is the role of the JDBC DriverManager class?

`DriverManager` 就是“驱动调度中心”：

- 管理已加载的 JDBC drivers
- 根据 URL 选择合适 driver
- 提供 `getConnection(...)` 建立连接