Basics

1. Introduction to Java

Java is a high-level programming language developed by **James Gosling** at Sun Microsystems in 1995. It follows the principle of **"Write Once, Run Anywhere" (WORA)**, meaning Java programs can run on any platform that has a Java Virtual Machine (JVM). It is widely used for building enterprise applications, Android apps, and web applications.

Key Features

- **Object-Oriented**: Based on objects and classes.
- **Platform-Independent**: Runs on JVM, making it OS-agnostic.
- **Secure**: No explicit pointers; includes runtime checks.
- **Multithreaded**: Supports concurrent execution of multiple tasks.

Example

Here's a simple program to print "Hello, World!" in Java:

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

Interview Tips

- **What makes Java platform-independent?**
    - Bytecode is the key! Java source code is compiled into platform-neutral bytecode, which can be executed on any JVM.
- **Where is Java used?**

  - Java powers web apps (Spring Framework), Android apps, financial systems, and large-scale enterprise software.

---

2. History of Java

Java's journey started in 1991 as a project named **Oak**, intended for consumer electronics. It was renamed to **Java** in 1995 due to trademark issues.

Important Milestones

- **1995**: Java 1.0 released with basic object-oriented features.
- **2004**: Java 5 introduced generics, annotations, and enhanced for-loops.
- **2014**: Java 8 revolutionized Java programming with lambda expressions and Streams API.
- **2021**: Java 17 introduced sealed classes and other advanced features.

Fun Fact

The name "Java" came from the developers' love for Java coffee.

Interview Tips

- **Who created Java?**
  - James Gosling.
- **What was Java originally called?**
  - Oak, inspired by an oak tree outside Gosling's office.

---

3. Java Versions

Java evolves with each version, introducing features that simplify coding and improve performance.

Version	Release Date	Key Features
JDK 1.0	January 1996	Initial release with basic object-oriented features.
JDK 1.1	February 1997	Event handling model, inner classes, JavaBeans, JDBC.

J2SE 1.2	December 1998	Swing API, Collections framework, JIT compiler.
J2SE 1.3	May 2000	HotSpot JVM, RMI enhancements, JavaSound.
J2SE 1.4	February 2002	assert keyword, NIO, Logging API, XML processing.
J2SE 5.0	September 2004	Generics, metadata (annotations), enumerated types, enhanced for loop.
Java SE 6	December 2006	Scripting language support, improvements to Web Services, JDBC 4.0.
Java SE 7	July 2011	Diamond operator, try-with-resources, NIO.2, Fork/Join framework.
Java SE 8	March 2014	Lambda expressions, Stream API, new Date-Time API, Nashorn JavaScript engine.
Java SE 9	September 2017	Module system (Project Jigsaw), JShell, HTTP/2 client.
Java SE 10	March 2018	Local-variable type inference (var keyword).
Java SE 11	September 2018	Long-Term Support (LTS), new HTTP client, local-variable syntax for lambda parameters.
Java SE 12	March 2019	Switch expressions (preview), JVM constants API.
Java SE 13	September 2019	Text blocks (preview), dynamic CDS archives.
Java SE 14	March 2020	Switch expressions, records (preview), pattern matching for instanceof (preview).
Java SE 15	September 2020	Text blocks, sealed classes (preview), hidden classes.
Java SE 16	March 2021	Records, pattern matching for instanceof, vector API (incubator).
Java SE 17	September 2021	LTS, sealed classes, pattern matching for switch (preview).

Java SE 18	March 2022	Simple web server, UTF-8 by default, code snippets in API documentation.
Java SE 19	September 2022	Virtual threads (preview), structured concurrency (incubator).
Java SE 20	March 2023	Scoped values (incubator), record patterns (preview).
Java SE 21	September 2023	LTS, pattern matching for switch, sequenced collections.
Java SE 22	March 2024	Enhanced pattern matching, string templates.
Java SE 23	September 2024	Universal generics, asynchronous stack traces.

Interview Tips

- **Why is Java 8 so important?**
    - Lambda expressions introduced functional programming concepts, making Java more concise and readable.
- **What are sealed classes?**
    - A feature in Java 17 that restricts inheritance, ensuring only specific classes can extend a sealed class.

---

4. Identifiers

Identifiers are the names you give to variables, methods, or classes. They help make the code meaningful and readable.

Rules

1. Must start with a letter, `$`, or `_`.
2. Cannot use reserved keywords.
3. Case-sensitive and must be meaningful.

Example

```
int age = 25;     // 'age' is an identifier
String $name = "John";
```

Interview Tips

- **What are valid identifiers?**
    - Valid: `age`, `_count`, `$value`.
    - Invalid: `1name`, `int` (keyword).
- **Can an identifier start with a number?**
    - No, identifiers cannot begin with numbers in Java.

---

5. Keywords

Keywords are reserved words that Java uses for its syntax and cannot be used for identifiers.

Examples

- `class`, `int`, `void`, `static`, `if`, `else`

Code Example

```
public class Example {
    public static void main(String[] args) {
        int number = 10; // 'int' is a keyword
    }
}
```

Interview Tips

- **Can you use keywords as identifiers?**

- No, Java reserved keywords cannot be used as variable or method names.

---

6. Variables

Variables are containers that store data in a program. Java supports three types:

1. **Local Variables**: Declared inside a method or block.
2. **Instance Variables**: Belong to a specific object.
3. **Static Variables**: Shared among all instances of a class.

Code Example

```
public class VariablesExample {
    static int staticVar = 10; // Shared among all objects
    int instanceVar = 20;      // Unique to each object

    public void display() {
        int localVar = 30;     // Exists only during method e
xecution
        System.out.println(localVar);
    }
}
```

Interview Tips

- **Difference between static and instance variables?**
    - **Static**: Stored in the method area, shared across objects.
    - **Instance**: Stored in heap memory, unique to each object.
- **Where are local variables stored?**
    - In the stack memory.

---

7. Data Types

Data types specify the type of data that a variable can hold.

Primitive Types

- **int**: Whole numbers (4 bytes)
- **float**: Decimal numbers (4 bytes)
- **char**: Single characters (2 bytes, Unicode)
- **boolean**: True/False (1 bit)

Non-Primitive Types

- **String**: Sequence of characters.
- **Array**: Collection of elements.
- **Objects**: Instances of classes.

Example

```
int age = 25;
String name = "Java";
```

Interview Tips

- **Why is String non-primitive?**
  - String is a class in Java, allowing methods like `.length()` and `.concat()`.
- **What is the size of `char` in Java?**
  - 2 bytes, as Java uses Unicode.

---

8. Wrapper Classes

Wrapper classes convert primitive types into objects, allowing them to be used in collections like `ArrayList`.

Examples

- `int → Integer`
- `boolean → Boolean`

Code Example

```
public class WrapperExample {
    public static void main(String[] args) {
        int num = 5;
        Integer wrappedNum = num; // Autoboxing
        int unwrappedNum = wrappedNum; // Unboxing
    }
}
```

Interview Tips

- **What is autoboxing?**
    - Automatic conversion of primitive to wrapper (e.g., `int → Integer`).
- **Why are wrapper classes important?**
    - Collections like `ArrayList` can only store objects, not primitives.