# Java Programming & Environment

## What is Java?

High-level, robust, OO, secure language & platform.

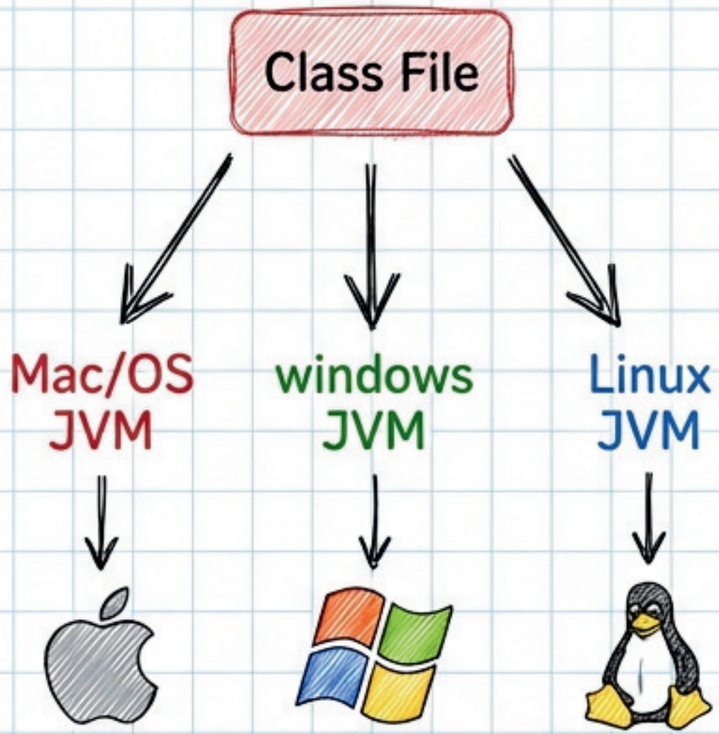
## History:

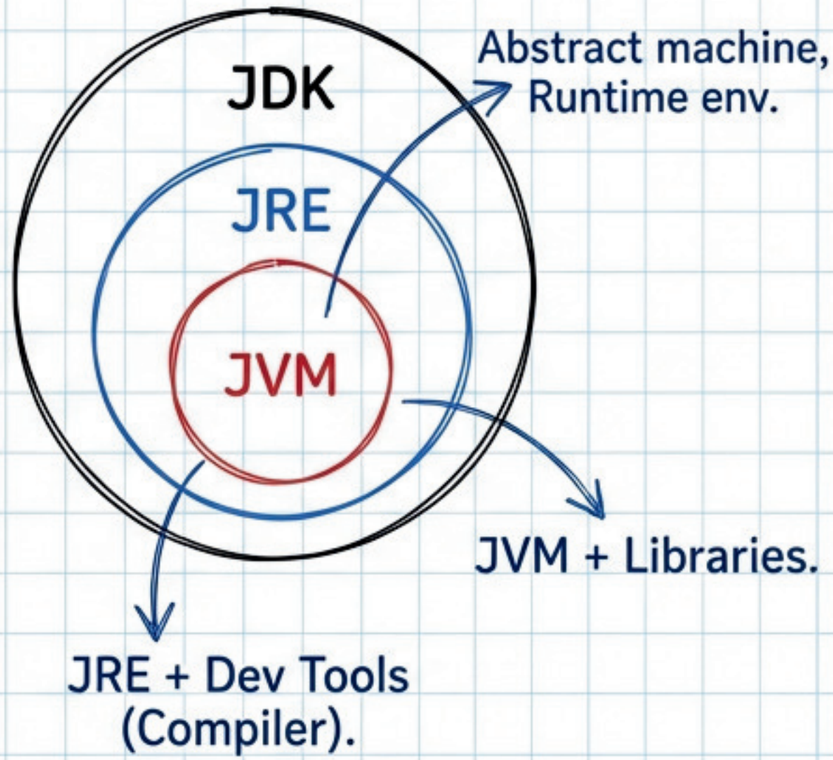
Sun Microsystems (1995).
Father: James Gosling.

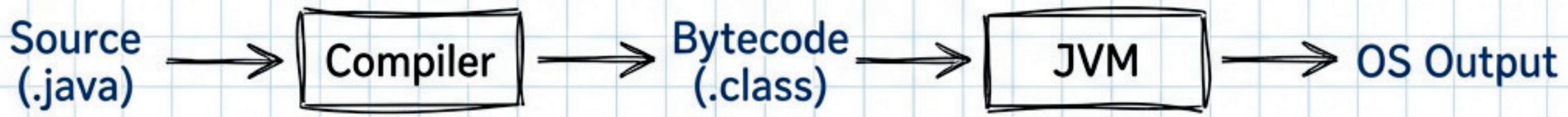
## Features:

- ★ Simple (no pointers)
- ★ Object-Oriented
- ★ Platform Independent (WORA)
- ★ Secured (Sandbox)
- ★ Robust (GC)
- ★ Portable
- ★ Multi-threaded

## JVM Architecture



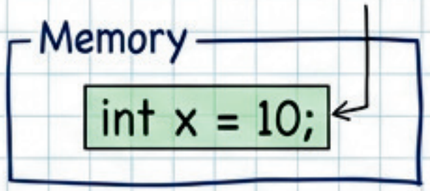




notes by ~ Kamal Kishor

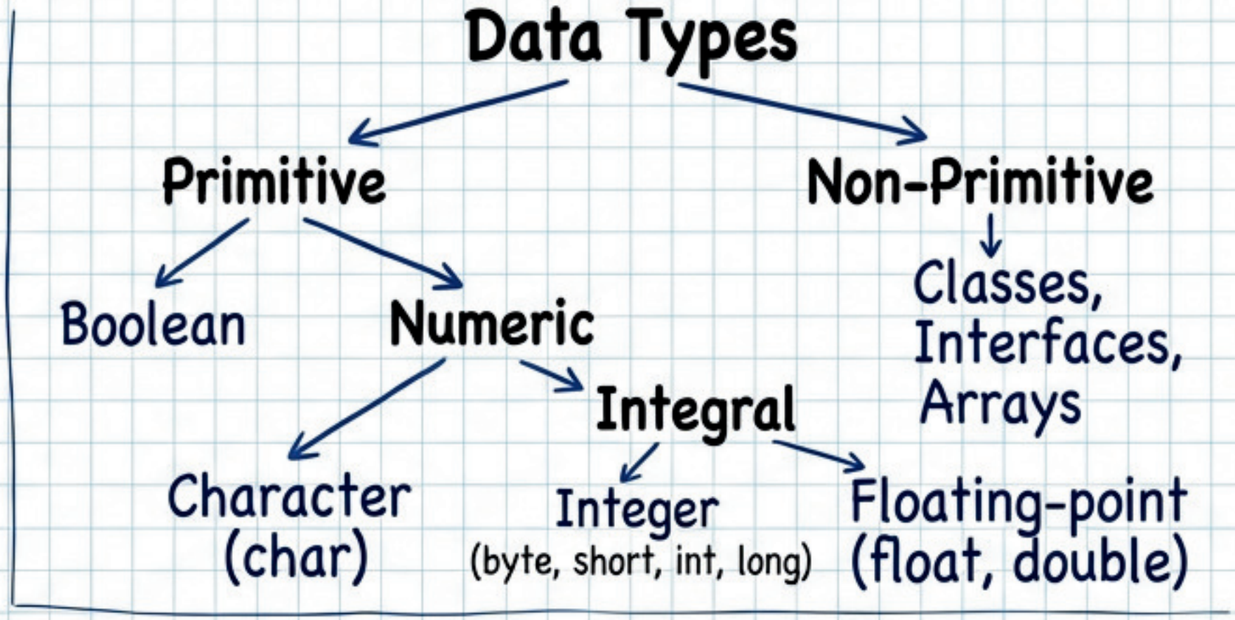
# Basics: Variables & Data Types

## Variables (Containers)

**Definition:** Name of memory location.




- **Local:** Inside method.
- **Instance:** Inside class, outside method.
- **Static:** Single copy, shared.




**Unicode System:** Java uses 2 bytes (supports intlL languages). Range: \u0000 to \uFFFF.

## Keywords:

Reserved words (class, public, void, etc.). Cannot be used as identifiers.

| Unary | Arithmetic (+, -, *, /) | Shift (<<, >>) | Relational (<, >, <=, >=, ==, !=) | Bitwise | Logical (&&, ||, !) | Ternary (? :) | Assignment (=) |
|---|---|---|---|---|---|---|---|

# Control Flow Statements

## 1. Decision Making

**if statement:**
condition check.

**if-else:**
true/false block.

**switch:** multiple cases.

efficient for many options

`if(a>b)`

`switch(choice)`

## 2. Loops (Iteration)

**for loop:**
fixed iterations.

**while loop:**
unknown iterations.

**do-while:**
executes at least once
(exit controlled).

**for-each:**
for(Type var : array){}

## 3. Jump

**break:** stops execution.

**continue:**
skips to next iteration.

## First Program

Blueprint

Entry point (static saves memory)

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

Output

margin notes

Margin notes

# OOP Concepts (The Big Picture)

## Definitions

- **Object:** Entity with state & behavior (e.g., Chair).
- **Class:** Blueprint/Collection of objects.

```
class Car {
   int speed;
}
Car c = new Car();
```

margin notes

Marain noles

## Inheritance

IS-A relationship.
Parent-Child. Reusability.

## Polymorphism

One task, different ways.
Overloading/Overriding.

## Abstraction

Hiding internal details.
Showing functionality.

## Encapsulation

Wrapping code & data.
Data hiding.

## Design Principles

**Coupling:** Dependency. Goal → Loose Coupling | **Cohesion:** Focus. Goal → High Cohesion
**Association:** Relation. | **Aggregation:** Weak HAS-A. | **Composition:** Strong HAS-A

# Class Structure & Keywords

## Constructors

- Init object. No return type. Same name as class.
- Types: Default, Parameterized.

Private Constructor: Prevents object creation.
Used in Singleton Pattern.

### Constructor vs Method

| | | |
|---|---|---|
| 1. Init state of object. | 1 | 1. Expose behavior of object. |
| 2. Must not have return type. | 2 | 2. Must have return type. |
| 3. Invoked Implicitly. | 3 | 3. Invoked explicitly. |
| 4. Java compiler provides default if none exist. | 4 | 4. Not provided by compiler in any case. |
| 5. Name must be same as class name. | 5 | 5. Name may or may not be same as class name. |

## Keywords

**Static** - Belongs to Class (memory management).

- Variable (Common property)
- Method (No object needed)
- Block (Runs before main)

**This** - Refers to current object.

1. Current instance variable (this.var)
2. Invoke current method (this.method())
3. Invoke current constructor (this())
4. Pass as argument in method (method(this))
5. Pass as argument in constructor (Constructor(this))
6. Return current class instance (return this)

# Inheritance & Polymorphism

## Inheritance (IS-A)

- One object acquires properties of another.

Types:
- Single
- Multilevel
- Hierarchical

```
class A {
}
class B extends A {

}
```

Note: Multiple inheritance NOT supported (ambiguity).

## Polymorphism

**Method Overloading** (Compile-time):
- Same name, different params.

```
add(int a, int b)
add(int a, int b, int c)
```

**Method Overriding** (Runtime):
- Subclass specific implementation.
  - ☑ Same name
  - ☑ Same params
  - ☑ IS-A relationship
- Covariant Return Type: Return type varies in subclass direction.

## super

1. Immediate parent variable.
2. Parent method.
3. Parent constructor.

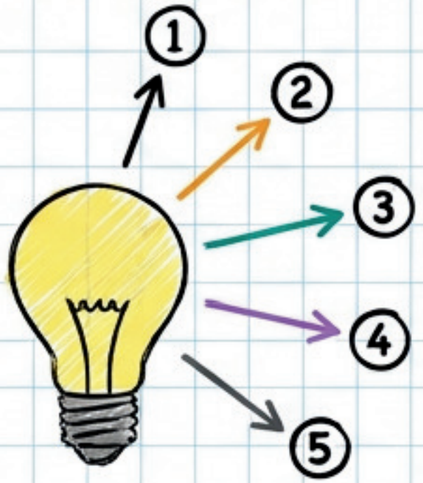
# Abstraction & Interfaces

**Final Keyword**
- Variable → Constant
- Method → No Override
- Class → No Inheritance

**Runtime Poly:**
Upcasting (Parent ref → Child obj)

## Abstract Class (0-100%): abstract

- Can have abstract & non-abstract methods
- No instantiation
- Must be declared abstract
- Cannot be instantiated
- Can have final methods

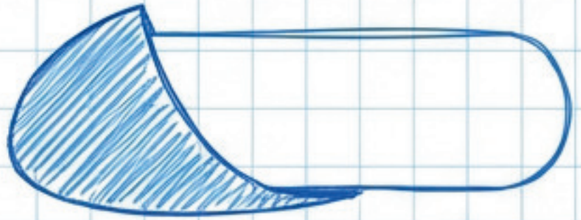


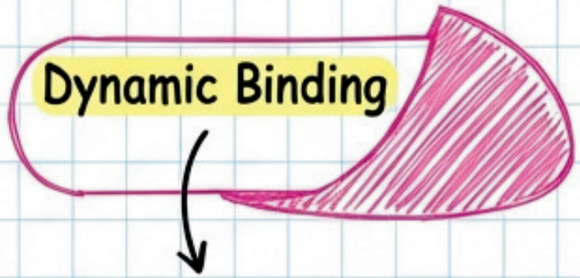

## Interface (100% Abstract) : implements

- Only static constants & abstract methods
- Supports Multiple Inheritance

| Abstract | vs | Interface |
|---|---|---|
| extends | | implements |
| partial abstraction | | full abstraction |
| no multiple inheritance | | supports multiple inheritance |

**Static vs Dynamic Binding**






(Object type determined at runtime).

Abstract Method:

```
abstract void show();
```

Interface A:

```
interface A {
        void show();
}
```

# Encapsulation & Organization

## Packages

Group of similar classes/interfaces.

- Built-in: java.lang, java.util
- User-defined: `package mypack;`

## Encapsulation

Wrapping code/data in a capsule.

Data Hiding: private data members + public getters/setters.

Benefits: Control (read-only/write-only).

```
class Student {
    private int marks;
}
```

Strict

### Visibility

| Modifier | Class | Package | Subclass | World |
|---|---|---|---|---|
| Private: | Y | N | N | N |
| Default: | Y | Y | N | N |
| Protected: | Y | Y | Y | N |
| Public: | Y | Y | Y | Y |

# Arrays, Object Class & Utilities

**Arrays:** Object, fixed-size, similar data types.

```
int a[] = {1,2,3};          (1D)
int a[][] = new int[2][2]; (2D)
```

Pros: Random access.
Cons: Size limit.

## Utilities

Wrapper Classes: Primitive <-> Object.

Autoboxing: int → Integer

Unboxing: Integer → int

**Object Class:** Mother of all classes.

- toString()
- equals()
- finalize()
- clone()
- wait()

Cloning: Exact copy.
`Cloneable` interface.

Strictfp: Consistent floating-point results.

Math Class: min, max, sin, cos.

# String Handling

## Basics

String: Sequence of chars. Immutable (Cannot be modified).

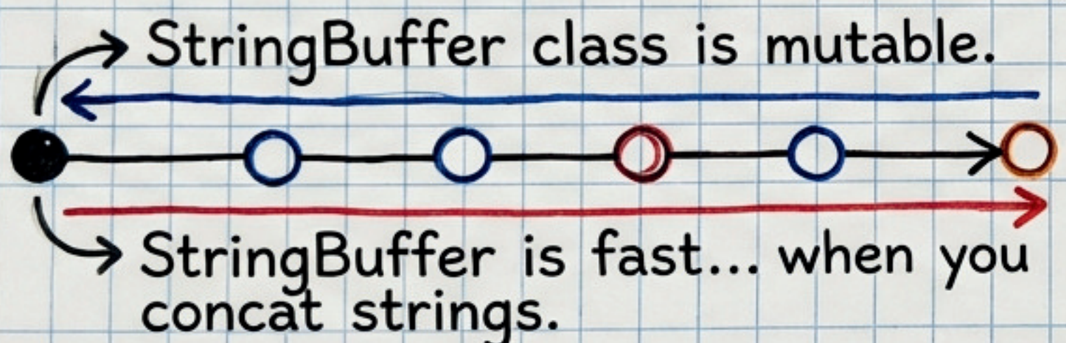
String s = "Java"; (Literal)
new String()

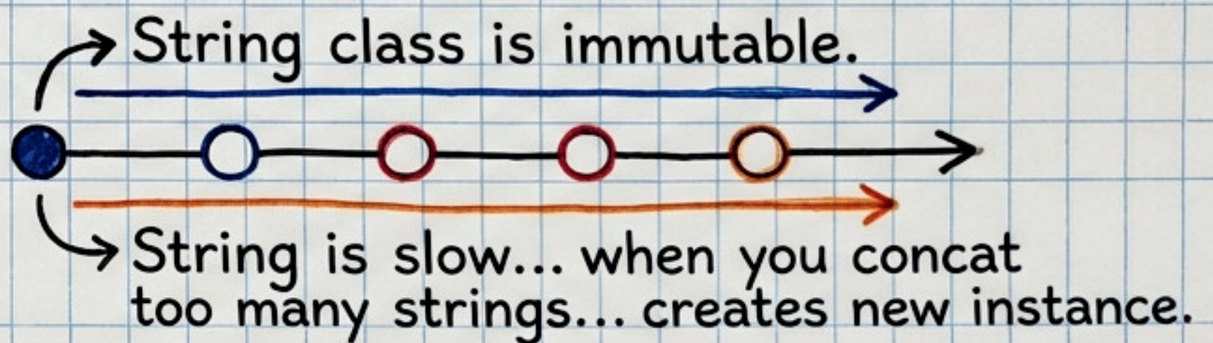
## StringBuffer

- Mutable.
- Synchronized (Thread-safe).
- Slow.

## StringBuilder

- Mutable.
- Non-Synchronized.
- Fast.



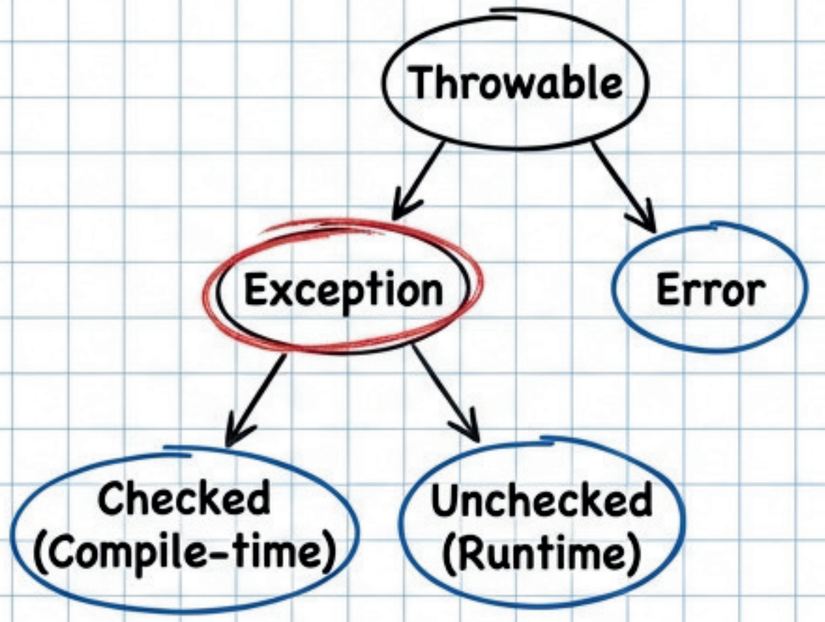

StringTokenizer: Legacy class for breaking strings.

toString(): String representation of object.

# Exception Handling

## Basics:

- **Exception**: Runtime error disrupting flow.




```
try {

} catch(Exception e) {
}
```

## Keywords:

- **try**: Risky code.
- **catch**: Handle code.
- **finally**: Always executed (Cleanup).

| Throw | Throws |
|---|---|
| Explicitly throw an exception inside method. | Declare an exception in method signature. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Followed by an instance. | Followed by class. |
| Within the method. | With the method signature. |
| Cannot throw multiple exceptions. | Can declare multiple exceptions. e.g. `public void method() throws IOException, SQLException'. |

## Final vs Finally vs Finalize:

| Final | Finally | Finalize |
|---|---|---|
| Constant (Keyword). Cannot be modified, overridden, or inherited. | Block (Always runs). Executes important code whether exception occurs or not (cleanup). | Method (Cleanup before GC). Performed just before object is garbage collected. |

# Multithreading (Basics)

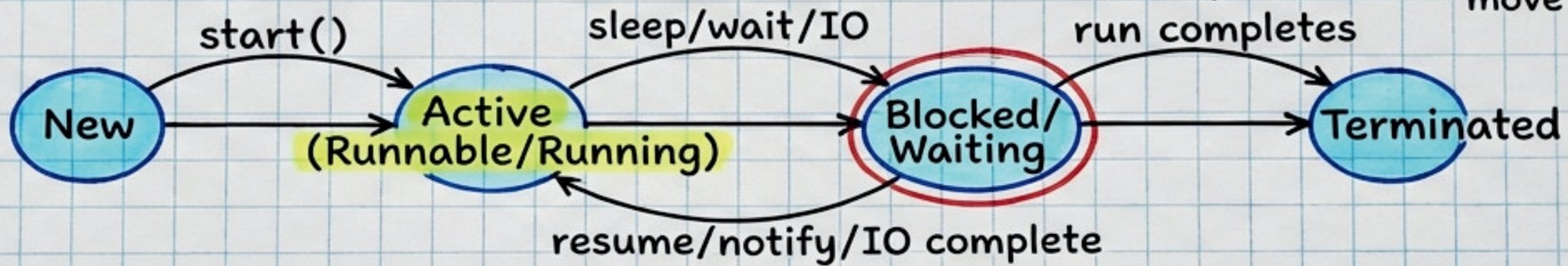
- **Basics:**
  - **Thread:** Lightweight sub-process.
  - **Creation:**
    1. Extend Thread class.  2. Implement Runnable.

```
class MyThread extends Thread {
    public void run() {}
}
```

Thread-safe is important!

Synchronized

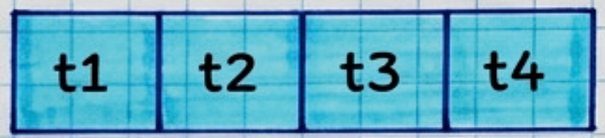
Sharing resources in use the goal of instruct to move completed.



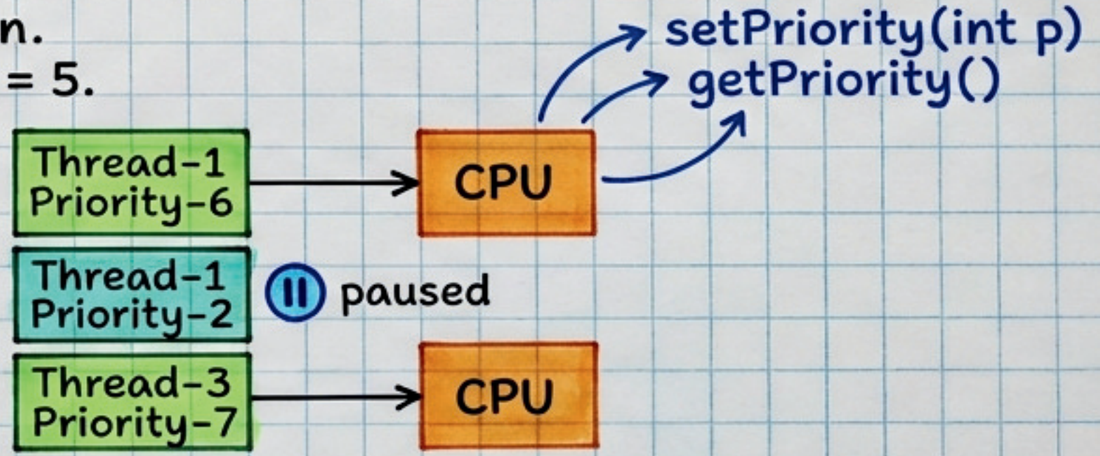

## Thread Scheduler & Priority

- Thread Scheduler: Decides execution.
- Priority: 1 (Min) to 10 (Max). Norm = 5.
- Algorithms:

Thread-safe is insurra- to hand+- important!




First Come First Serve Scheduling

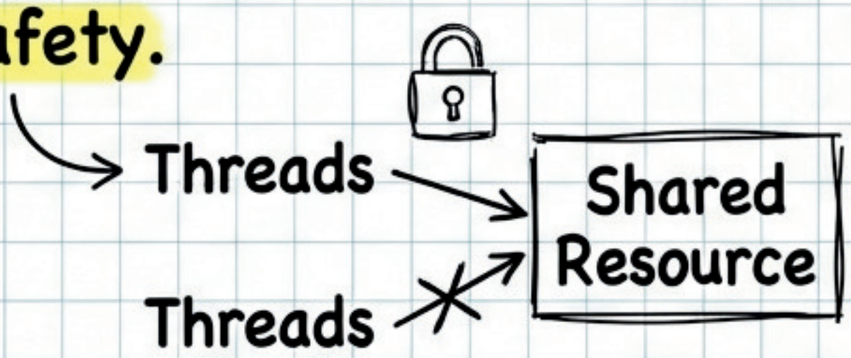



Preemptive-Priority Scheduling

# Multithreading (Advanced) & GC

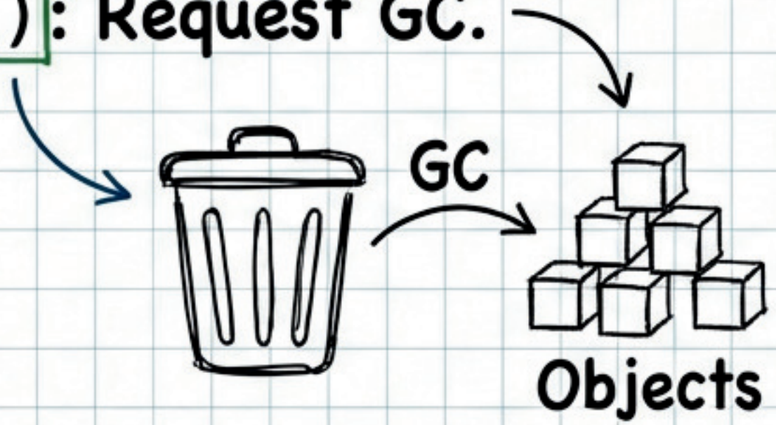
## Methods & Concepts

- `sleep(ms)`: Pause execution.
- `join()`: Wait for thread to die.
- **Daemon Thread**: Service provider (e.g., GC). Dies with user threads.
- **Thread Pool**: Reusable worker threads.
- **Synchronization**: Lock mechanism, thread safety.




## Garbage Collection

- Reclaims unused memory.
- **Unreferenced by**: Nulling, Reassigning.
- `finalize()`: Called before GC.
- `gc()`: Request GC.



# Collections Framework

Collection

List | Set | Queue

## List

Ordered, Duplicates OK.

- ArrayList (Dynamic array)
- LinkedList (Double linked)
- Vector (Sync, legacy)

ArrayList vs Vector

1. Not Synced
2. Not Legacy
3. 50% resize

1. Synced (Thread-safe)
2. Legacy
3. Double resize

Vector is thread-safe but slow!

## Set

Unique elements.

- HashSet (No order)
- LinkedHashSet (Insertion order)
- TreeSet (Sorted)

Set
↑
AbstractSet
↑
HashSet


## Map

Key-Value pairs.

- HashMap (1 null key)
- TreeMap (Sorted keys)

Map
↓
AbstractMap
↓
HashMap
↓
LinkedHashMap

**Queue/Deque**: FIFO.

**Sorting**: Comparable (single) vs Comparator (multiple).

TreeSet & TreeMap use natural ordering or comparator.

# Advanced: JDBC, Reflection & Applets

## JDBC

Java Database Connectivity.


Register Driver → Types: Bridge, Native, Network, Thin.

Get Connection

Create Statement

Execute Query

Close Connection

## Reflection & Applets

- Reflection: Inspect/Modify runtime behavior.
- Applets: Run in browser.

Applet Lifecycle:

1. init()
2. start()
3. paint()
4. stop()
5. destroy()