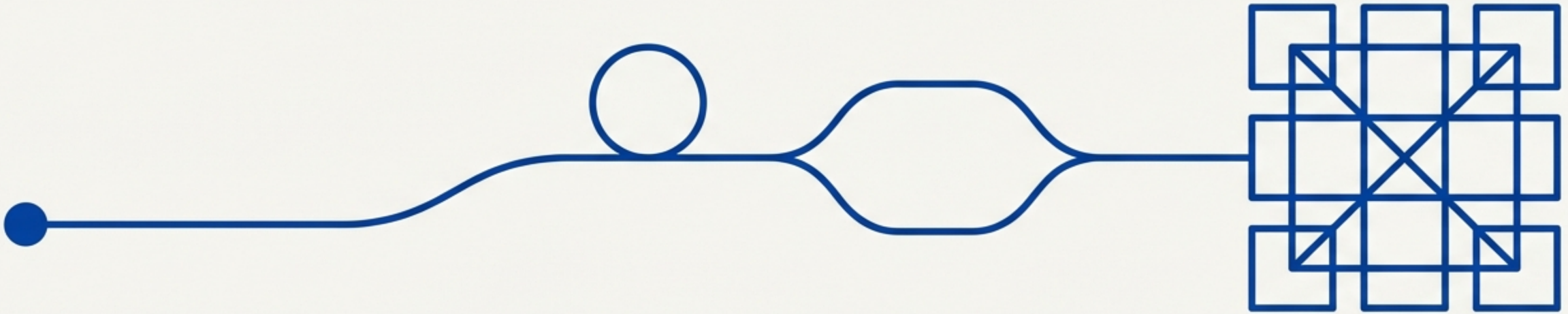
FROM FIRST LINE TO FULL APPLICATION: A JOURNEY THROUGH CORE JAVA



A structured guide to mastering the fundamentals of the Java programming language.

UNDERSTANDING THE JAVA ECOSYSTEM

A Java program begins as human-readable source code (`.java`) and is transformed into machine-executable code through a two-step process.

JDK (Java Development Kit):
The software tools required to *develop* Java applications. Includes the compiler (`javac`).

JRE (Java Runtime Environment):
The software platform required to *run* Java programs. Includes the JVM.

JVM (Java Virtual Machine):
A software machine that executes Java bytecode, translating it into machine-specific code for the local platform.

YOUR FIRST PROGRAM: THE ANATOMY OF `main()`

```java
// FirstProgram.java

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

The `main()` method is the entry point for any Java application. It is always defined with the header `public static void main(String args[])`.

- **`public`**: The method can be accessed from outside the class.
- **`static`**: It can be called without creating an object of the class (e.g., `ClassName.main()`). The JVM calls it directly.
- **`void`**: The method does not return any value.
- **`String args[]`**: An array of `String` objects that stores command-line arguments passed to the program. This parameter is compulsory.

THE BUILDING BLOCKS: PRIMITIVE DATA TYPES & BASIC I/O

Primitive Data Types

Java provides a set of fixed-size primitive types for storing data efficiently.

- **Integer Types**: `byte` (1), `short` (2), `int` (4), `long` (8 bytes)
- **Floating-Point Types**: `float` (4), `double` (8 bytes). Default is `double`; use a suffix for float: `float x = 23.45f;`
- **Character Type**: `char` (2 bytes, Unicode)
- **Boolean Type**: `boolean` (`true` or `false`)

Basic I/O Operations

Input

Use the `Scanner` class from the `java.util` package.

```
import java.util.Scanner;
Scanner sc = new Scanner(System.in);
int myInt = sc.nextInt();
String myString = sc.next();
```

Output

Use `System.out.print()` and `System.out.println()`. The + operator performs string concatenation.

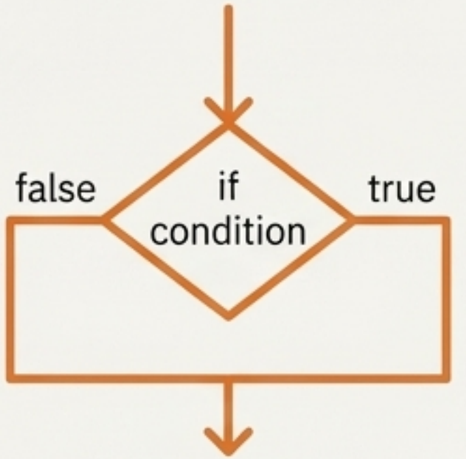
```
System.out.println("Answer = " + myInt);
```

EXPRESSING LOGIC: OPERATORS AND CONTROL STATEMENTS

Operators

A quick overview of Java's operators for performing calculations and comparisons.

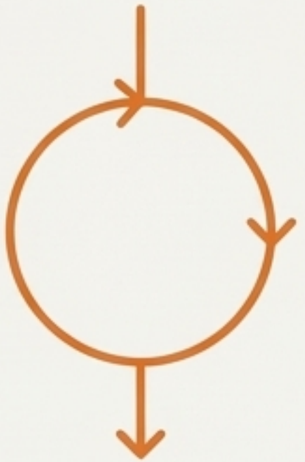
- **Arithmetic**: `+`, `-`, `*`, `/`, `%`, `++`, `--`
- **Relational**: `<`, `>`, `<=`, `>=`, `==`, `!=`
- **Logical**: `&&` (AND), `||` (OR), `!` (NOT)
- **Ternary**: `?:` (e.g., `condition ? value_if_true : value_if_false`)



Control Flow Statements

Direct the execution path of your program based on conditions and loops.

- **Selection**: `if-else` for conditional branching.
- **Iteration**: `for`, `while`, and `do-while` for looping.
- **Case Control**: `switch-case` for multi-way branching.
- **Branching**: `break` and `continue` to alter loop behaviour.



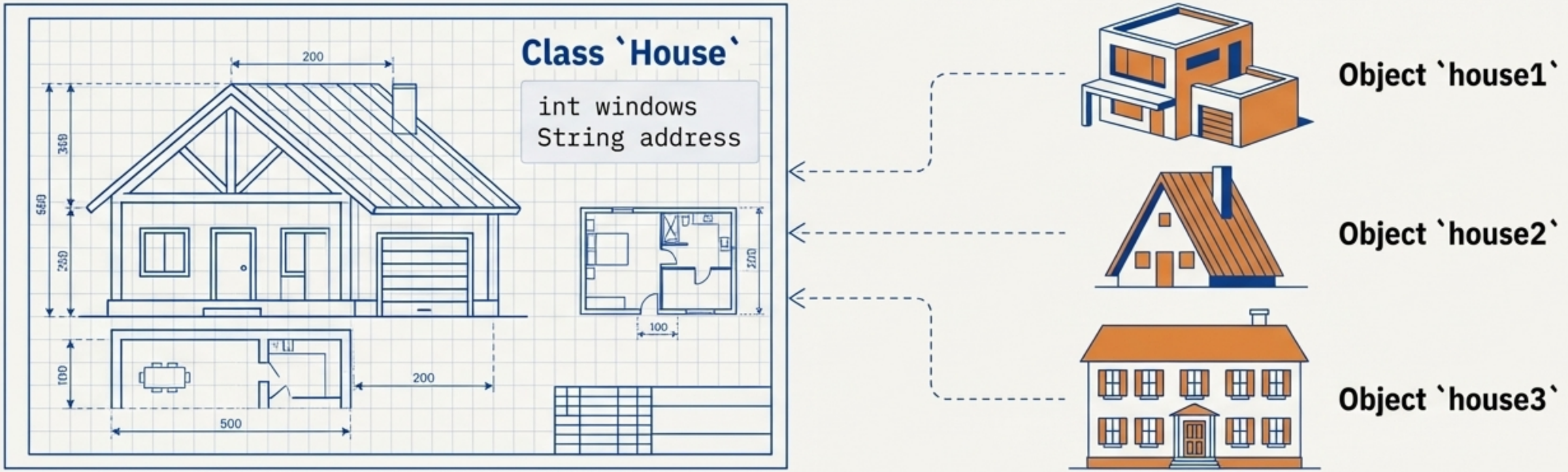
THE BLUEPRINT: UNDERSTANDING CLASSES, OBJECTS, AND ENCAPSULATION

Core Concept

A **class** is a blueprint or prototype that defines the variables (attributes) and methods (behaviours) for a particular type of object. An **object** is an instance of a class.

Encapsulation

The practice of bundling an object's data (variables) and the methods that operate on that data into a single unit (the class). Access to the data is controlled through **access specifiers** (`public`, `protected`, `private`), a principle also known as data hiding.



BRINGING OBJECTS TO LIFE: CONSTRUCTORS AND THE `this` REFERENCE

Creating Objects

Memory is allocated and the object is initialised in a single statement.

```
ClassName obj = new ClassName(<args>);
```

- **new**: An operator that allocates memory for a new object.
- **Constructor()**: A special method that initialises the newly created object.

Types of Constructors

A class can have multiple (overloaded) constructors.

- **Default**: Assigns default initial values (e.g., `x = 0;`).
- **Parameterised**: Assigns user-specified values (e.g., `Data(int a)`).
- **Copy**: Copies values from an existing object (e.g., `Data(Data d)`).

The `this` Reference

The `this` keyword is an implicit reference to the current object. It's used to distinguish between instance variables and parameters with the same name.

```
public Data(int x) {
   this.x = x; // 'this.x' is the instance variable
}
```

ORGANISING YOUR CODEBASE WITH PACKAGES

Packages are containers for grouping related classes, preventing naming conflicts and improving reusability and organisation.

How it Works

1. **Declaration**: At the top of your source file, declare the package.

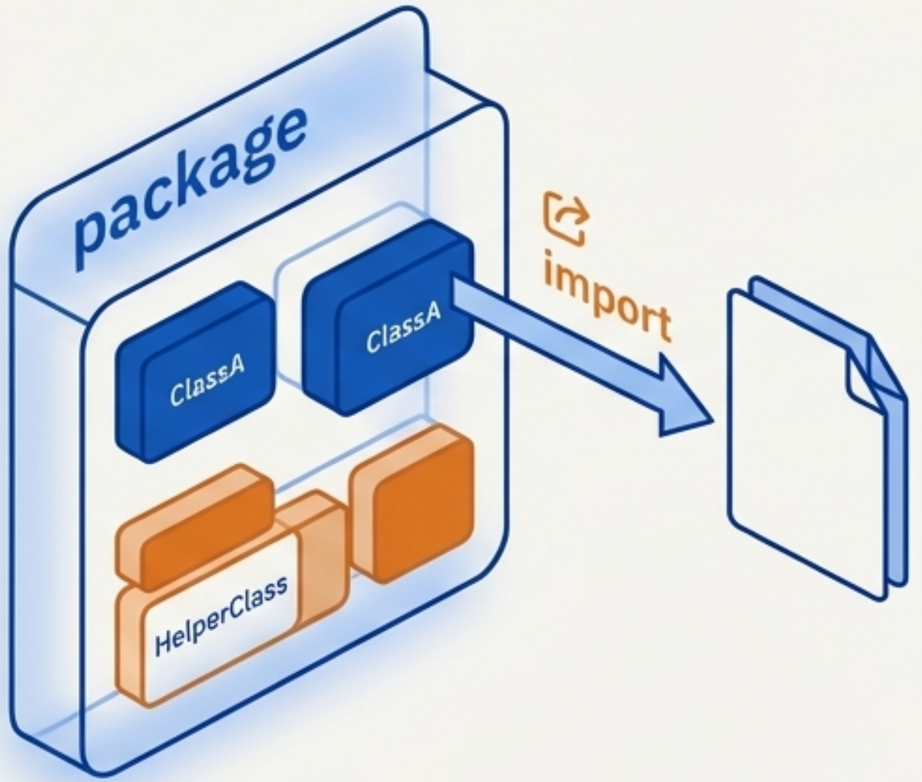
```
package mypackage;
```

2. **Compilation**: Compile the class, placing the .class file in a corresponding directory structure. The `-d .` flag automates this.

```
javac -d . MyClass.java
```

3. **Import**: In another program, use the import statement to access the class.

```
import mypackage.MyClass;
or
import mypackage.*;
```



Key Built-in Packages

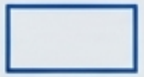
- `java.lang`: Core language classes (`String`, `Integer`, `System`). Imported automatically.
- `java.util`: Utility classes like `Scanner` and `Vector`.
- `java.io`: Classes for input/output operations.
- `java.net`: Classes for network programming.

COMPILE-TIME POLYMORPHISM: THE POWER OF METHOD OVERLOADING

Method overloading allows a class to have more than one method with the same name, as long as their parameter lists are different (either by number of parameters or their data types). This is resolved at compile time.

Practical Example

A single `area()` method for different shapes.



```
class Geometry {
    // Area of a rectangle
    public static float area(float length, float breadth) {
        return length * breadth;
    }

    // Area of a circle
    public static float area(float radius) {
        return 3.142f * radius * radius;
    }
}
```

Usage

The correct method is called based on the arguments provided.

```
System.out.println("Rectangle area = " + Geometry.area(4, 5));
System.out.println("Circle area = " + Geometry.area(4));
```

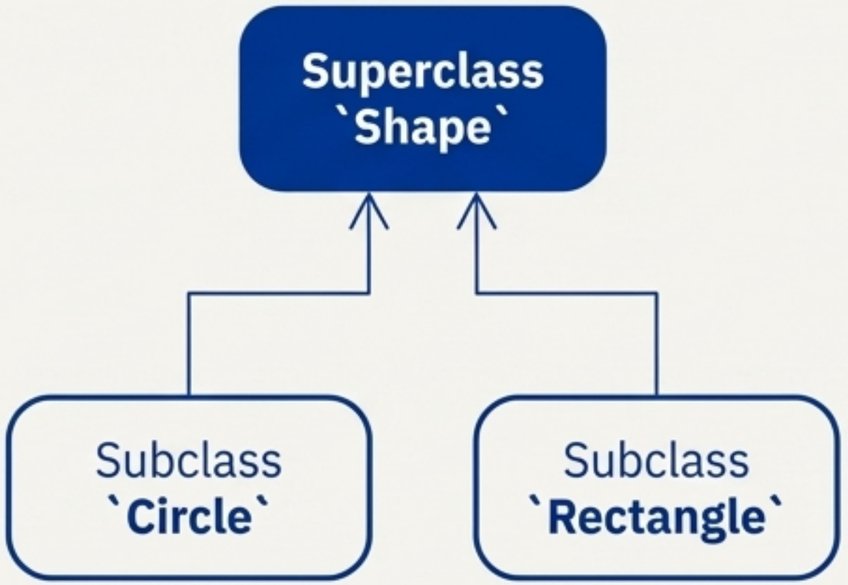
Outcome

This provides different responses from what is logically the same method name, enhancing code clarity.

BUILDING ON FOUNDATIONS: INHERITANCE AND ACCESS CONTROL

Inheritance allows a new class (subclass) to acquire the properties and methods of an existing class (superclass). This is achieved using the `extends` keyword.

```
class SubClassName extends SuperClassName {
  ...
}
```



Access Specifiers and Visibility

- `private`: Accessible only within the owner class. Not inherited.
- `default` (no modifier): Accessible within the same package.
- `protected`: Accessible within the package and by subclasses in any package.
- `public`: Accessible from anywhere.

DYNAMIC BEHAVIOUR: RUNTIME POLYMORPHISM WITH METHOD OVERRIDING

Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The method name, parameters, and return type must be identical.

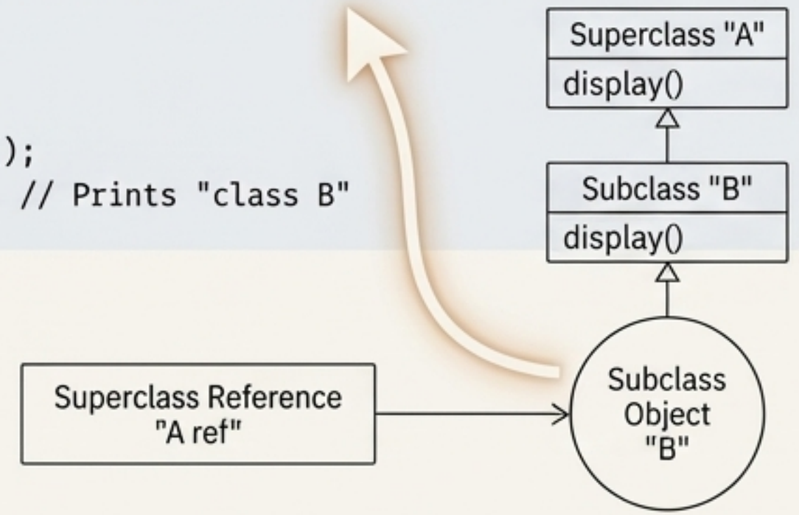
Key Concepts

- **Dynamic Binding:** The decision of which method to execute is made at runtime, based on the object's actual type.
- **Superclass Reference:** A superclass reference can hold a subclass object. `Shape s = new Circle();`
- The **`super` Keyword:** From within a subclass, use `super.methodName()` to explicitly call the superclass's version of an overridden method.

- **Overloading**: Same method name, different parameters. Compile-time.
- **Overriding**: Same method name, same parameters (in subclass). Runtime.

```
class A {
  public void display() { System.out.println("class A"); }
}
class B extends A {
  @Override // Annotation for clarity
  public void display() { System.out.println("class B"); }
}

// In main():
A ref = new B();
ref.display(); // Prints "class B"
```



DEFINING CONTRACTS: ABSTRACT CLASSES AND INTERFACES

Both are used to achieve abstraction, defining a set of methods that subclasses must implement. They cannot be instantiated directly.

Abstract Class

- A class declared with the `abstract` keyword.
- Can contain both `abstract` methods (declarations only) and concrete methods (with implementations).
- A class must be abstract if it has one or more abstract methods.
- Use `extends` for inheritance.

```
abstract class Shape {
  abstract void area();
}
```

Interface

- A completely abstract ‘class’ that can only contain abstract method declarations and `public static final` constants.
- A class uses the `implements` keyword to implement an interface. It must provide an implementation for all methods.
- A class can implement multiple interfaces, allowing Java to support a form of multiple inheritance.

```
interface Shape {
  void area();
}
class Circle implements Shape { ... }
```

BUILDING RESILIENT APPLICATIONS WITH EXCEPTION HANDLING

What is an Exception?

A runtime error that disrupts the normal flow of a program. Java represents exceptions as objects thrown at runtime.

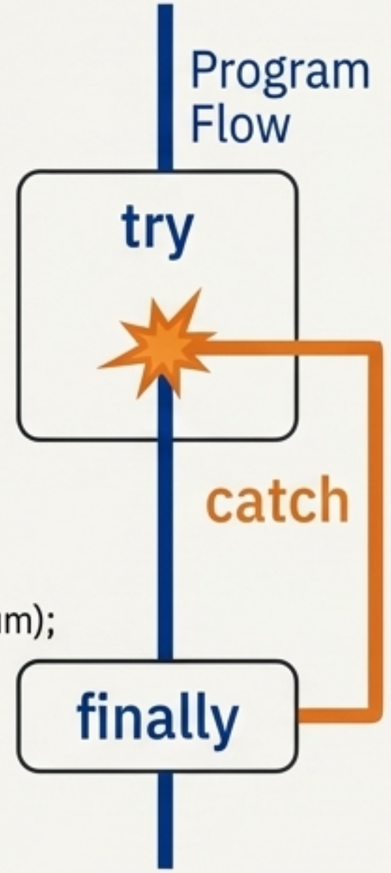
The Handling Mechanism: `try-catch-finally`

- **`try`**: Enclose the code that might throw an exception.
- **`catch`**: Catches and handles a specific type of exception. You can have multiple **`catch`** blocks for one **`try`**, ordered from most specific to most general.
- **`finally`**: A block of code that is always executed, whether an exception is thrown or not. Used for cleanup tasks like closing files.

Code Example

Preventing a crash from an invalid array access.

```
int a[] = {10, 20, 30};
int sum = 0;
try {
   for (int i=0; i<5; i++) { // This will cause an error
      sum += a[i];
   }
} catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Error: Array access out of bounds.");
   System.out.println("Partial sum calculated: " + sum);
} finally {
   System.out.println("End of operation.");
}
}
```



ACHIEVING CONCURRENCY WITH MULTITHREADING

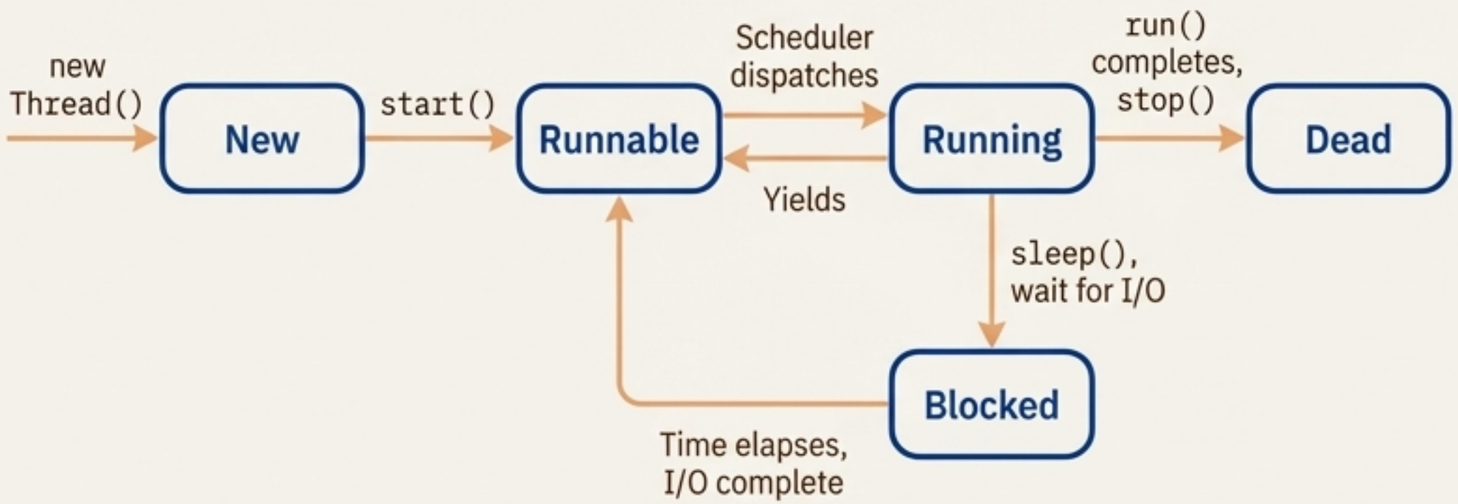
What is a Thread? A single, active flow of execution within a program. Multithreading allows a program to perform multiple tasks concurrently, improving performance and responsiveness.

Creating a Thread

1. Define a class that extends `java.lang.Thread`.
2. Override the `run()` method with the code the thread should execute.
3. Create an instance of your thread class and call its `start()` method. The `start()` method internally calls your `run()` method.

The Thread Lifecycle

A thread moves between several states during its lifetime.

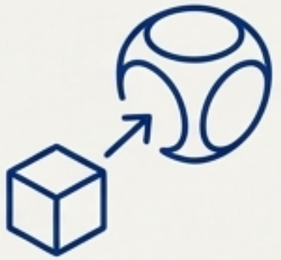


YOUR CORE JAVA TOOLKIT: ESSENTIAL API CLASSES

Mastering Core Java involves understanding not just the syntax but also the powerful classes provided by the standard library. Here are some of the most fundamental tools at your disposal.

`String`

The class for handling sequences of characters. Crucially, `String` objects are *immutable* in Java. For mutable strings, use `StringBuffer`.

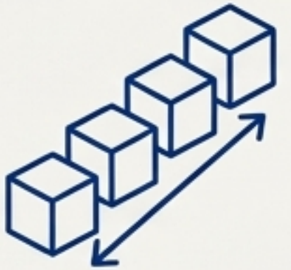


Wrapper Classes

(`Integer`, `Double`, `Boolean`): Object representations of primitive types. They provide utility methods like `Integer.parseInt()` to convert strings to numbers.

`Math`

A class containing `static` methods for common mathematical operations, such as `Math.sqrt()`, `Math.pow()`, and `Math.max()`.



`Vector`

A classic collection class from `java.util` that provides a dynamic, resizable array of objects.

`Object`

The ultimate superclass of all classes in Java. It provides fundamental methods like `toString()` and `equals()`.