Programming Languages

- **Java**
- **C**
- **C++**
- **Python**
- **Go**

Problem Solving in Programming

- **Business Problem (Domain)**
  - Solve problems related to specific business domains (Banking, Travel, Insurance, etc.).

Approaches in Software Development

1. **Data-Centric Approach**
   - **App → DB → CRUD** (Create, Read, Update, Delete)
2. **Object-Centric Approach**
   - **App → Object → DB → CRUD**

Advantages & Concepts

- **Ripple Effect**
  - A small change can have widespread impacts (like a ripple in a pond).
- **Software Entropy**
  - Over time, software becomes more disordered and harder to maintain.
- **Design Trade-offs**
  - **Generic Design** → Takes more time but is flexible.
  - **Specific Design** → Faster but less adaptable.
- **Balance** → Achieved using **Design Patterns**.

Java vs. C Compilation

Java	C
`Welcome.java` (Source)	`Welcome.c` (Source)
`Welcome.class` (Bytecode) → **Platform Independent** (WORA*)	`Welcome.obj` → `Welcome.exe` → **Platform Dependent**
JRE → Runtime Environment (Not Platform Independent)	
JDK → JRE + Compiler	

*WORA = Write Once, Run Anywhere

Class & Object Concepts

- **Class** → Template/Blueprint (e.g., `Employee`).
- **Object** → Instance of a class (e.g., `Employee e = new Employee();`).

Business Domain Modeling

- **Business Objects** (Nouns) → Customer, Account, Loan, Card.
  - Example:

```
class Customer {
    String name;
    String email;
    String address;
}
```

- **Business Services** (Actions) → LoanService, CreditCardService.
  - Example:

```
class HomeLoanService {
    void apply();
    void close();
    void enquire();
}
```

Requirements & Use Cases

- **Requirement** → **Use Case** → **Business Objects & Services**.

- **Business Domain** defines:
  - **Concepts** → Business Objects.
  - **Capabilities** → Business Services (Coarse/Fine-grained).

4 Pillars of OOP

1. **Inheritance (Inh)** → Reuse and extend classes.
2. **Polymorphism (Poly)** → One interface, multiple forms.
3. **Encapsulation (Encap)** → Bundling data + methods (like a capsule).
4. **Abstraction (Abst)** → Hiding complex implementation.

Design for Change

- Example:

```
class Car {
    // Properties & methods
}
```

Key Keywords in Java

- `null` → No object reference.
- `this` → Refers to current object.
- `super` → Refers to parent class.
- **Data Members** → Should be `private` for encapsulation.

Constructors

- A special method with the **same name as the class**.
- Used for **object initialization**.

Static Keyword

- Associated with the **class**, not objects.
- Used for **shared properties/methods** across instances.

Java Memory Management: Garbage Collection, Heap & Stack

1. Garbage Collection (GC)

- **Purpose**: Automatically reclaims unused memory by destroying unreachable objects.
- **How it works**:
  - Identifies objects no longer referenced by the program.
  - Runs in the background via the **Garbage Collector** (part of JVM).
- **Key Methods**:
  - `System.gc()` – Suggests JVM to run GC (not guaranteed).

2. Heap Memory

- **Purpose**: Stores **objects and runtime data** (allocated via `new` keyword).
- **Key Points**:
  - GC runs here to free unused memory.
  - **OutOfMemoryError** if heap is full.

3. Stack Memory

- **Purpose**: Stores **method calls, local variables, and references**.
  - **StackOverflowError** if stack is full (e.g., infinite recursion).

Heap vs. Stack Comparison

Feature	Heap	Stack
Storage	Objects & instance variables	Method calls & local variables
Access	Slower (dynamic allocation)	Faster (fixed memory)
Scope	Global (shared)	Thread-specific
Errors	`OutOfMemoryError`	`StackOverflowError`