**Domain-Driven Design (DDD) in C# - বাংলায় ব্যাখ্যা**

Domain-Driven Design (DDD) এমন একটি software development approach, যেটা জটিল business সমস্যাকে বুঝে তার solution তৈরি করতে সাহায্য করে। DDD মূলত business logic এবং domain model-কে কেন্দ্র করে software তৈরি করার পদ্ধতি। নিচে C# ভাষায় DDD-এর মৌলিক ধারণাগুলো বাংলায় ব্যাখ্যা করছি।

**🧠 ১. Domain Driven Design-এর মূল লক্ষ্য কী?**

DDD-এর প্রধান উদ্দেশ্য হচ্ছে **business domain**-কে ভালভাবে বোঝা এবং সেটাকে software-এ reflect করা। মানে হচ্ছে, যে business সমস্যা সমাধান করতে চাই, তার নিয়ম-কানুন, behavior, terminology সব কিছু যেন কোডে ফুটে উঠে।

**🧩 ২. DDD-এর গুরুত্বপূর্ণ অংশ**

**✅ ২.১. Domain**

Domain মানে হচ্ছে business-এর কাজের ক্ষেত্র। যেমন: eCommerce, Banking, Healthcare, ইত্যাদি। Domain হলো problem space।

**✅ ২.২. Model**

Model হলো domain-এর simplified abstraction। এটিই কোডে represent করা হয়।

**✅ ২.৩. Ubiquitous Language**

Developer এবং business expert সবাই মিলে যে common language ব্যবহার করে সেটা। Domain model বানাতে এই ভাষা ব্যবহার হয়।

**🏗️ ৩. DDD-এর Building Blocks (C# কোড সহ ব্যাখ্যা)**

**🔸 ৩.১. Entity**

Entity এমন একটি object যার একটি identity থাকে এবং সময়ের সাথে এর state পরিবর্তন হতে পারে।

public class Customer

{

public Guid Id { get; private set; }

public string Name { get; private set; }

public Customer(Guid id, string name)

{

Id = id;

Name = name;

}

public void ChangeName(string newName)

{

Name = newName;

}

}

**🔸 ৩.২. Value Object**

Value Object-এর identity থাকে না। এগুলো immutable হয়।

public class Address

{

public string Street { get; }

public string City { get; }

public Address(string street, string city)

{

Street = street;

City = city;

}

}

**🔸 ৩.৩. Aggregate & Aggregate Root**

Aggregate হলো একগুচ্ছ Entity ও Value Object-এর সমন্বয়, যার মধ্যে শুধু Aggregate Root-ই বাইরের সাথে যোগাযোগ করে।

public class Order : AggregateRoot

{

public Guid Id { get; private set; }

private List<OrderItem> \_items;

public void AddItem(Product product, int quantity)

{

\_items.Add(new OrderItem(product, quantity));

}

}

**🔸 ৩.৪. Repository**

Repository pattern ব্যবহৃত হয় database থেকে Entity আনতে ও রাখতে।

public interface ICustomerRepository

{

Customer GetById(Guid id);

void Add(Customer customer);

}

**🔸 ৩.৫. Domain Service**

যদি কোন logic এমন হয় যা কোন একটি Entity-র মধ্যে যায় না, তখন তা Domain Service-এ রাখা হয়।

public class PaymentService

{

public void ProcessPayment(Order order)

{

// Payment logic here

}

}

**🧪 ৪. DDD প্রয়োগ করার সময় যা মাথায় রাখতে হয়**

1. **Business logic যেন domain model-এর ভেতর থাকে**, না যে কোন সেবার মধ্যে।
2. Infrastructure (DB, API) যেন Domain Layer-এর উপর নির্ভর না করে।
3. ভালোভাবে Ubiquitous Language গঠন করতে হবে।
4. Domain Model সবসময় Clean Architecture অনুসরণ করবে।

**🧱 ৫. সাধারণ Layer Structure (C# DDD প্রজেক্ট)**

- Domain

- Entities

- ValueObjects

- Services

- Interfaces

- Application

- UseCases

- Infrastructure

- Repositories

- DataAccess (EF Core etc.)

- Presentation

- Controllers / APIs

**📘 উপসংহার**

Domain-Driven Design (DDD) এমন একটি শক্তিশালী software design pattern যা code structure এবং business logic-এর মধ্যে গভীর সংযোগ স্থাপন করে। যদি আপনি বড়, জটিল প্রজেক্ট নিয়ে কাজ করেন, তাহলে DDD আপনাকে maintainable ও scalable architecture গঠন করতে সাহায্য করবে।

চলুন আরও বিস্তারিতভাবে **Domain-Driven Design (DDD)** ব্যাখ্যা করি — **C#-এ কিভাবে এটি বাস্তবায়ন করা যায়, তার Bengali ব্যাখ্যা সহ**।

**🧠 DDD-এর মূল ধারণা (Core Philosophy)**

DDD বলে, software design শুরু হওয়া উচিত **business domain** বোঝা থেকে, শুধু CRUD নয়। মানে, আপনি কেবলমাত্র database-এর field আর API বানানোর চিন্তা করবেন না — বরং, কীভাবে ব্যবসায়িক নিয়ম (business rules) কাজ করে, সেই অনুযায়ী একটা rich model তৈরি করবেন।

**🏛 DDD-এর স্তরভিত্তিক معماري (Layered Architecture)**

DDD-তে একটি typical application কে ৪টা বড় স্তরে ভাগ করা হয়:

**১. Domain Layer (সবচেয়ে গুরুত্বপূর্ণ)**

* **Entities**
* **Value Objects**
* **Domain Services**
* **Business Rules**

**২. Application Layer**

* **Use Cases** (যেমন: RegisterUser, PlaceOrder)
* Application এর প্রবাহ (workflow) নিয়ন্ত্রণ করে

**৩. Infrastructure Layer**

* Database, External API, File System ইত্যাদি

**৪. Presentation Layer**

* API Controllers, UI Layer, CLI, ইত্যাদি

**🔍 DDD-এর প্রধান উপাদানগুলোর বিস্তারিত আলোচনা (With C# Example)**

**✅ ১. Entity**

Entity এমন একটি object যার নিজের একটি identity থাকে।

public class Product

{

public Guid Id { get; private set; }

public string Name { get; private set; }

public Product(Guid id, string name)

{

Id = id;

Name = name;

}

public void Rename(string newName)

{

Name = newName;

}

}

🔹 Entity-র প্রধান বৈশিষ্ট্য: Entity কখনো নাম বা বৈশিষ্ট্য পরিবর্তন করলেও তার Id একই থাকে।

**✅ ২. Value Object**

এগুলো data encapsulate করে, কিন্তু এদের নিজস্ব identity থাকে না।

public class Money

{

public decimal Amount { get; }

public string Currency { get; }

public Money(decimal amount, string currency)

{

Amount = amount;

Currency = currency;

}

}

🔹 Value Object immutable হয় — মানে তৈরি করার পর আর পরিবর্তন হয় না।

**✅ ৩. Aggregate & Aggregate Root**

**Aggregate:** একটি Entity ও তার সাথে সম্পর্কিত Value Object-দের একটা গ্রুপ।  
**Aggregate Root:** Aggregate-এর "entry point"। শুধু এটি public হয়, বাকি সব encapsulated হয়।

public class Order : AggregateRoot

{

public Guid Id { get; private set; }

public List<OrderItem> Items { get; private set; } = new();

public void AddItem(Product product, int quantity)

{

Items.Add(new OrderItem(product, quantity));

}

}

🔹 Order হলো Aggregate Root, আর OrderItem হলো ভিতরের অংশ।

**✅ ৪. Repository (Database Access Interface)**

Domain Model কখনো নিজে DB-তে save বা fetch করে না। Repository handle করে।

public interface IOrderRepository

{

Order GetById(Guid id);

void Save(Order order);

}

🔹 Infrastructure Layer এ এই Interface-এর implementation থাকবে।

**✅ ৫. Domain Service**

কখনো কখনো এমন logic থাকে যা কোনও একক Entity-তে যায় না, তখন তা Domain Service-এ রাখা হয়।

public class DiscountService

{

public decimal CalculateDiscount(Customer customer)

{

if (customer.IsPremium) return 0.20m;

return 0.05m;

}

}

**✅ ৬. Application Service (Use Case)**

Use Case বাস্তবায়নের জন্য Application Layer ব্যবহার করা হয়।

public class PlaceOrderService

{

private readonly IOrderRepository \_orderRepo;

public PlaceOrderService(IOrderRepository orderRepo)

{

\_orderRepo = orderRepo;

}

public void Execute(PlaceOrderCommand command)

{

var order = new Order(command.CustomerId);

order.AddItem(command.Product, command.Quantity);

\_orderRepo.Save(order);

}

}

**🧱 C#-এ DDD Project Structure উদাহরণ**

📁 /src

├── 📁 Domain

│ ├── Entities

│ ├── ValueObjects

│ ├── Interfaces

│ └── Services

├── 📁 Application

│ ├── Commands

│ ├── Services

├── 📁 Infrastructure

│ ├── Data

│ └── Repository

├── 📁 Presentation

│ └── Controllers (ASP.NET Core API)

**🧪 DDD অনুসরণ করতে গিয়ে প্রায় দেখা যায় এমন কিছু ভুল**

| **ভুল** | **সঠিক** |
| --- | --- |
| Domain Model-এ কোনো logic না রেখে শুধু property রাখার প্রবণতা | Domain Model এর মধ্যেই business rule রাখতে হবে |
| Controller-এর মধ্যে অনেক logic লেখা | Logic থাকবে Application বা Domain Service-এ |
| Entity ও Value Object না বুঝে সব কিছু Entity বানানো | Proper design principle অনুযায়ী ভাগ করতে হবে |
| Entity কে সরাসরি save করা বা update করা | Repository pattern ব্যবহার করতে হবে |

**📚 আপনি যা চাইতে পারেন পরবর্তী ধাপে**

1. ✅ একটি বাস্তব উদাহরণ (যেমন: E-commerce Order Management DDD architecture সহ)
2. ✅ Clean Architecture + DDD Integration
3. ✅ CQRS (Command Query Responsibility Segregation) DDD-তে কীভাবে যুক্ত হয়
4. ✅ Unit Testing Domain Layer