**DRY Principle in C#**

**What is DRY?**

**DRY (Don't Repeat Yourself)** is a fundamental software development principle that encourages **code reusability** and **reduces redundancy**.  
💡 **Goal:** Write code **once** and **reuse** it instead of duplicating logic.

**Why Follow DRY?**

✅ Easier Maintenance  
✅ Reduces Bugs & Errors  
✅ Improves Readability  
✅ Enhances Scalability

**How to Implement DRY in C#?**

**🔹 1️ Use Methods (Avoid Code Duplication)**

Instead of repeating the same logic, move it into a **method**.

❌ **Without DRY (Code Duplication)**

csharp

CopyEdit

Console.WriteLine("Hello, Alice!");

Console.WriteLine("Hello, Bob!");

Console.WriteLine("Hello, Charlie!");

✅ **With DRY (Using a Method)**

csharp

CopyEdit

void Greet(string name)

{

Console.WriteLine($"Hello, {name}!");

}

Greet("Alice");

Greet("Bob");

Greet("Charlie");

**🔹 2️ Use Classes & Objects (Encapsulation)**

❌ **Without DRY**

csharp

CopyEdit

string emp1Name = "John";

double emp1Salary = 50000;

string emp2Name = "Sarah";

double emp2Salary = 60000;

✅ **With DRY (Using a Class)**

csharp

CopyEdit

class Employee

{

public string Name { get; set; }

public double Salary { get; set; }

}

Employee emp1 = new Employee { Name = "John", Salary = 50000 };

Employee emp2 = new Employee { Name = "Sarah", Salary = 60000 };

**🔹 3️ Use Inheritance (Avoid Repeating Code in Multiple Classes)**

❌ **Without DRY (Duplicated Code)**

csharp

CopyEdit

class Dog

{

public void Speak() => Console.WriteLine("Bark!");

}

class Cat

{

public void Speak() => Console.WriteLine("Meow!");

}

✅ **With DRY (Using Inheritance)**

csharp

CopyEdit

class Animal

{

public virtual void Speak() => Console.WriteLine("Animal Sound");

}

class Dog : Animal

{

public override void Speak() => Console.WriteLine("Bark!");

}

class Cat : Animal

{

public override void Speak() => Console.WriteLine("Meow!");

}

**🔹 4️ Use Interfaces (Common Behavior Across Classes)**

✅ **With DRY (Using Interface)**

csharp

CopyEdit

interface IShape

{

double Area();

}

class Circle : IShape

{

public double Radius { get; set; }

public double Area() => Math.PI \* Radius \* Radius;

}

class Square : IShape

{

public double Side { get; set; }

public double Area() => Side \* Side;

}

💡 Now, both Circle and Square follow a common structure without duplication.

**🔹 5️ Use Generics (Avoid Code Rewriting)**

❌ **Without DRY (Rewriting Methods for Different Types)**

csharp

CopyEdit

void PrintInt(int value) => Console.WriteLine(value);

void PrintString(string value) => Console.WriteLine(value);

✅ **With DRY (Using Generics)**

csharp

CopyEdit

void Print<T>(T value) => Console.WriteLine(value);

Print(100); // Works for int

Print("Hello"); // Works for string

**🔹 6️ Use Design Patterns (Reusable Solutions)**

* **Singleton** (Ensures a single instance)
* **Factory Pattern** (Creates objects dynamically)
* **Repository Pattern** (Encapsulates data access logic)

✅ **Example: Singleton Pattern**

csharp

CopyEdit

class Singleton

{

private static Singleton instance;

private Singleton() { }

public static Singleton GetInstance()

{

if (instance == null)

instance = new Singleton();

return instance;

}

}

💡 **Prevents redundant object creation** by ensuring **only one instance**.

**🚀 Key Takeaways**

✅ **DRY = Avoid Code Duplication, Improve Reusability**  
✅ Use **Methods, Classes, Inheritance, Interfaces**  
✅ Apply **Generics & Design Patterns**  
✅ **Easier Maintenance & Scalability**