1. Classes & Objects

class is a blueprint, and an **object** is an instance of a class.

```
class Car
{
   public string brand;
}

Car myCar = new Car();
myCar.brand = "Toyota";
Console.WriteLine(myCar.brand);  // Output: Toyota
```

2. Value Types vs. Reference Types

- **Value Types** (ex. int, float, bool) store data **directly in memory** (stack).
- **Reference Types** (ex. class, string, arrays) store **a reference (memory address) to the data** (heap).

3. Type Casting

Converting a **variable from one type to another**.

- **Implicit Casting (Automatic):** When there is **no data loss**.

  ```
  int num = 10;
  double d = num; // No data loss
  ```

- **Explicit Casting (Manual):** When there **might be data loss**.

  ```
  double x = 9.8;
  int y = Convert.Toint16(x); // Decimal part removed, y = 9
  ```

4. String Split & Join

- **Split:** Break a string into an array.
- **Join:** Convert an array back into a string.

Ex.

```
string text = "apple,banana,grape";
string[] words = text.Split(',');  // ["apple", "banana", "grape"]

string joined = string.Join(" - ", words);
Console.WriteLine(joined); // Output: apple - banana - grape
```

5. StringBuilder

Modifying strings using string is inefficient since **strings are immutable**. StringBuilder improves performance by **modifying strings in place**.

Ex:

```
StringBuilder sb = new StringBuilder();
sb.Append("Hello ");
sb.Append("World!");
Console.WriteLine(sb.ToString());  // Output: Hello World!
```

6. Constructor

special method runs when an object is created.

```
class Car
{
    public string brand;

    public Car(string brandName)
    {
        brand = brandName;
    }
}

Car car1 = new Car("Honda");
Console.WriteLine(car1.brand);  // Output: Honda
```

7. Static Classes

A **class that cannot be instantiated** and contains only static members.

```
static class MathHelper
{
    public static int Add(int a, int b) => a + b;
}
Console.WriteLine(MathHelper.Add(5, 10)); // Output: 15
```

9. Abstract Classes & Interfaces

- **Abstract Classes**: Cannot be instantiated, used as a **base class**.
- **Interfaces**: Define **a contract** for classes to follow.

```
abstract class Animal
{
    public abstract void MakeSound();
}
```

SOME KEY WORDS

❖ Access Modifiers

كلمات تستخدم لتحديد مستوى الوصول إلى **المتغيرات، الخصائص، الدوال، أو classes** هذه المحددات تساعد في **إخفاء التفاصيل الداخلية للكود** وتعزيز **مبدأ التغليف (Encapsulation)**

Control access to class members (public, private, protected,internal).

```
class Person
{
    private int age; // Can't be accessed outside
    public string Name; // Can be accessed anywhere
}
```

❖ ref & out Keywords

- **ref**: Passes arguments **by reference**.
- **out**: Requires the variable to be assigned inside the method.

```
void ModifyValue(out int number)
{
    number = 20;
}
int num;
ModifyValue(out num);
Console.WriteLine(num);  // Output: 20
```

❖ Enums

allow you to define a **set of named constants**.

```
enum Days { Sunday, Monday, Tuesday }
Days today = Days.Monday;
```

❖ Abstract Classes & Interfaces

- **Abstract Classes**: Cannot be instantiated, used as a **base class**.
- **Interfaces**: Define **a contract** for classes to follow.

```
abstract class Animal
{
    public abstract void MakeSound();
}
```