1: What is Type Safety?

- The **compiler or runtime prevents you from using variables in ways that are not allowed by their type**.
- It ensures you **don't accidentally treat a string as a number**, or a boolean as an object, etc.

❖ it helps with
  - Avoiding bugs at runtime
  - Early detection of errors during compilation
  - Better code clarity and maintenance

❖ Ex. Languages
  - C#, Java, Rust, Swift

Languages that are type-safe enforce matching data types through static type checking or runtime checks

---

2: What Are Bitwise Operators?

Bitwise operators are special operators that perform operations **directly on the individual bits** (binary digits: 0 or 1) of integer values.

They work at the **bit level**, unlike normal arithmetic operators which work on whole numbers.

❖ **Common Bitwise Operators**

Operator	Name	Description
&	**AND**	**Returns 1 only if both bits are 1**
`	**`**	**OR**
^	**XOR (exclusive OR)**	**Returns 1 if only one of the bits is 1**
~	**NOT (bitwise complement)**	**Flips all bits (1 → 0, 0 → 1)**
<<	**Left Shift**	**Shifts bits to the left (adds zeros on the right)**
>>	**Right Shift**	**Shifts bits to the right (removes bits on the right)**

❖ **Use Cases of Bitwise Operators:**

- **Low-level programming**
- **Working with flags**
- **Encryption**
- **Performance-critical code**
- **Graphics programming**

3: **What Are checked and unchecked in C#?**

In C#, **checked** and **unchecked** are **keywords** that control how the program handles **arithmetic overflow** in integral types (like int, byte, long).

❖ **What's Arithmetic Overflow?**

When a number **exceeds the storage limit** of its data type.

Example:

```
byte x = 255;
x = (byte)(x + 1); // Overflow! 255 + 1 = 256, but byte can only hold 0–255
```

❖ **checked – Catches Overflow**

```
checked
{
   int a = int.MaxValue;
   int b = a + 1;  // Throws OverflowException
}
```

Use this when you **want to detect overflow** and make sure it doesn't silently cause bugs.

- ❖ **unchecked – Ignores Overflow**

unchecked

{

  int a = int.MaxValue;

  int b = a + 1; // No error! Result will wrap around to int.MinValue

}

Use this when you **don't care about overflow** and want performance or wrap-around behavior.

- ❖ **Without checked or unchecked?**

By default:

- **In debug mode** → overflow throws exception (like checked)
- **In release mode** → overflow is ignored (like unchecked)

- ❖ **You Can Use Them Inline Too:**

int result = checked(a + b);

int result = unchecked(a + b);

- ❖ **Why It Matters?**
  - Preventing **unexpected bugs**
  - Controlling **overflow behavior**
  - Making code safer or faster, depending on the situation

4: What is a Garbage Collector (GC) in C#?

The **Garbage Collector (GC)** is a **built-in feature in .NET** (and many other modern languages) that automatically **manages memory**. Its job is to **free up memory** that your program no longer uses.

❖ **Why Do We Need a Garbage Collector?**

When you create objects using code like:

MyClass obj = new MyClass();

The object is stored in **memory (heap)**. If you don't remove it when you're done, it stays there and wastes memory.

In languages like C++, you must manually delete objects.
In C#, the **Garbage Collector does it for you automatically.**

❖ **What Does the Garbage Collector Do?**

- **Finds objects** your code is no longer using.
- **Frees (clears)** the memory used by those objects.
- **Prevents memory leaks** and improves performance.

➢ **Example:**

```
void CreateUser()
{
   User u = new User(); // allocated in memory
   // When this method ends, 'u' is no longer used
   // GC will eventually remove it from memory
}
```

You don't need to manually delete u. The GC will remove it when it's **not reachable** (no variables pointing to it).

❖ **When Does GC Run?**

- **Automatically** when system needs memory
- You can also force it (not recommended) with:

```
GC.Collect();
```

➢ **Key Benefits:**

- No need to manually manage memory
- Prevents memory leaks
- Simplifies development

➢ **GC Doesn't:**

- Immediately delete objects as soon as you're done
- Manage resources like files, database connections, etc. (you must dispose them manually using using or Dispose())

📝 **Summary:**

Feature	Explanation
What it does	Frees unused objects from memory
Works on	Objects in the **heap**
Automatic?	Yes
When it runs	When memory is low or triggered by system
Your job	Just stop referencing objects you're done with