🔹 `computeIfAbsent(key, func)`

Purpose: Map me agar key **nahi hai**, to pehle key create karta hai aur uski value `func` se assign karta hai.
Phir **us value ko return** karta hai (chahe purani ho ya nayi).

👉 Example:

```java
prereqMap.computeIfAbsent(neighbor, k -> new HashSet<>()).add(node);
```

Step by step:

1. Agar `neighbor` key **nahi hai** map me →
   → `k -> new HashSet<>()` chalega → ek empty HashSet banega
   → map me entry add ho jaayegi → `neighbor = new HashSet<>()`

2. Phir `.add(node)` karega → us set me `node` add ho jaayega

3. Agar `neighbor` pehle se hai → to uska existing set le lega aur usme `node` add karega

✅ So yes — yeh **key absent ho to create karta hai, aur hamesha data add karta hai**.

🔹 `getOrDefault(key, defaultValue)`

Purpose: Map me agar key **present hai**, to uski value return karo;
agar **absent hai**, to `defaultValue` return karo (lekin map me add nahi karta).

👉 Example:

```java
prereqMap.get(neighbor).addAll(prereqMap.getOrDefault(node, new HashSet<>()));
```

Meaning:

- `neighbor` ka set mil gaya (guaranteed present hai).
- `node` ka set lene ki koshish — agar nahi hai, to ek **temporary empty set** mil gaya.
- Fir `addAll()` se `node` ke saare prerequisites `neighbor` me add ho gaye.

🧩 **In short summary:**

Method	When key not present	When key present	Adds to map?	Example use
`computeIfAbsent(k, func)`	Creates new value with `func`	Returns existing value	✅ Yes	Safe add karne ke liye
`getOrDefault(k, def)`	Returns `def`	Returns existing value	❌ No	Safe access karne ke liye

Tera one-line explanation 👇

> ✅ `computeIfAbsent` → "agar key nahi hai to banao aur use karo"
> ✅ `getOrDefault` → "agar key nahi hai to temporary default value do, map mat badlo"

🧠 Step 1: `Function<T, R>` kya hai?

`Function<T, R>` ek **built-in functional interface** hai
jo Java 8 se aaya hai (`java.util.function` package me).

Iska matlab:

> "Ek aisa function jo ek input le (type T) aur ek output de (type R)."

```java
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

💬 Simple words me:

Agar likho
`Function<Integer, String>`
to iska matlab hai:

> "Yeh function ek `Integer` input lega aur ek `String` return karega."

Without lambda (Anonymous Class version)

java Copy code

```java
Function<Integer, Set<Integer>> func = new Function<Integer, Set<Integer>>() {
    @Override
    public Set<Integer> apply(Integer k) {
        return new HashSet<>();
    }
};
```

Yahan tum `Function` **interface** ka ek **anonymous implementation** bana rahe ho.

> "Anonymous class" = a class without a name that implements an interface (or extends a class) and you create its object immediately.

Lambda version

Lambda version likhne ka shortcut hota hai:

java Copy code

```java
Function<Integer, Set<Integer>> func = k -> new HashSet<>();
```

Yahan Java automatically samajh jaata hai:

- Input type → `Integer`
- Return type → `Set<Integer>`
- Method → `apply()`

Internally yeh likhne ke barabar hai 👇

Lambda:

```java
map.computeIfAbsent(key, k -> new HashSet<>());
```

Without lambda (old-style):

```java
map.computeIfAbsent(key, new Function<Integer, Set<Integer>>() {
    @Override
    public Set<Integer> apply(Integer k) {
        return new HashSet<>();
    }
});
```

Both mean the same thing —
difference sirf itna hai ke lambda short form me likha gaya hai.

🧠 Summary

Concept	Type	Purpose
`Function<T, R>`	Interface	Represents a function taking T and returning R
`new Function<>() { ... }`	Anonymous class	Old-style implementation
`k -> new HashSet<>()`	Lambda	Short, modern version of same thing
Used in	`computeIfAbsent`, `map`, `stream.map()`, etc.	