Dynamische Bindung (auch **Late Binding** genannt) bedeutet, dass die Entscheidung, welche Methode aufgerufen wird, zur **Laufzeit** getroffen wird und nicht während der **Kompilierung**.

Einfach gesagt: Wenn du eine Methode in einer abgeleiteten Klasse überschreibst (override), entscheidet das Programm **erst während der Ausführung** (nicht vorher), welche Version der Methode aufgerufen wird – die in der Basisklasse oder die in der Kindklasse.

Beispiel:

Stell dir vor, du hast eine Basisklasse **Tier** und eine abgeleitete Klasse **Hund**. Beide haben eine Methode **geraeusch()**:

```
class Tier {
    public void geraeusch() {
        System.out.println("Das Tier macht ein Geräusch");
    }
}

class Hund extends Tier {
    @Override
    public void geraeusch() {
        System.out.println("Der Hund bellt");
    }
}
```

Wenn du jetzt ein **Tier**-Objekt erstellst und zur Laufzeit festlegst, dass es ein Hund ist, wird die Methode von **Hund** aufgerufen – **nicht die von Tier**, obwohl der Datentyp "Tier" ist. Das nennt man dynamische Bindung.

```
Tier meinHund = new Hund();
meinHund.geraeusch(); // Ausgabe: "Der Hund bellt"
```

Hier wird zur **Laufzeit** entschieden, dass die Methode `geraeusch()` der Klasse **Hund** aufgerufen wird, nicht die der Basisklasse **Tier**. Das ist dynamische Bindung.

Statische Bindung (auch **Early Binding** genannt) bedeutet, dass die Entscheidung, welche Methode oder Variable verwendet wird, **zur Kompilierzeit** getroffen wird, also bevor das Programm ausgeführt wird.

Bei der statischen Bindung wird der genaue Typ eines Objekts schon während des Kompilierens bekannt, und der Compiler entscheidet, welche Methode oder Variable aufgerufen wird. Im Gegensatz zur dynamischen Bindung geschieht die Entscheidung also **nicht zur Laufzeit**, sondern **frühzeitig**.

Beispiel:

In Java wird die statische Bindung typischerweise für:

- **statische Methoden**,
- **private Methoden** und
- **Methoden von final Klassen**

Hier ein einfaches Beispiel:

```
class Tier {
    public void machGeräusch() {
        System.out.println("Das Tier macht ein Geräusch");
    }

    public static void statischeMethode() {
        System.out.println("Statische Methode von Tier");
    }
}

class Hund extends Tier {
    @Override
    public void machGeräusch() {
        System.out.println("Der Hund bellt");
    }

    public static void statischeMethode() {
        System.out.println("Statische Methode von Hund");
    }
}

public class Main {
    public static void main(String[] args) {
        Tier tier = new Tier();
        Hund hund = new Hund();

        // Statische Bindung: Das Objekt ist zur Kompilierzeit bekannt
        tier.machGeräusch();  // Ausgabe: "Das Tier macht ein Geräusch"
        hund.machGeräusch();  // Ausgabe: "Der Hund bellt"

        // Statische Methode: Hier wird die Methode der Klasse aufgerufen, nicht de
        tier.statischeMethode(); // Ausgabe: "Statische Methode von Tier"
        hund.statischeMethode(); // Ausgabe: "Statische Methode von Hund"
    }
}
```

In diesem Beispiel:

1. **Nicht-statische Methode (`machGeräusch`)**: Das Verhalten wird zur **Laufzeit** entschieden (dynamische Bindung), da es abhängig vom Objekttyp ist.
2. **Statische Methode (`statischeMethode`)**: Hier findet **statische Bindung** statt, weil statische Methoden zur **Kompilierzeit** an die Klasse gebunden sind, nicht an das Objekt. Das heißt, welche Methode aufgerufen wird, wird schon während des Kompilierens entschieden.

Statische Bindung wird immer dann verwendet, wenn der genaue Typ zur Kompilierzeit bekannt ist und das Verhalten sich nicht ändern kann, z. B. bei **statischen** oder **finalen Methoden**.