Overriding Uses

Monday, September 8, 2025 4:16 PM

◇ Root of All Classes

In Java, **every class implicitly extends the Object class** from java.lang.
This means all classes automatically inherit methods like:

- toString()
- equals(Object obj)
- hashCode()
- getClass()

◇ What Happens in System.out.println(object)?

When we write:

System.out.println(ram) ;

Java **implicitly calls the toString() method** of the object.
Thus, it's equals to ,

System.out.println(ram.toString());

◇ Source Code of Object.toString()

Here's the original implementation from the JDK (simplified):

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- getClass().getName() → gives class name.
- hashCode() → gives unique hash in hex.
- Example output:
- `lesson09.Person@15db9742`

◇ Overriding toString()

We can **override** toString() to provide a **custom, meaningful output**.

```
// overriding
@Override  new *
public String toString() {
    return name +" of age " + age + " work as " + occupation ;
}
```

◇ Demonstration in main class

```java
// object
Person ram = new Person( name: "Ram" , age: 42 , occupation: "Sales Man") ;

// output
System.out.println(ram);
```

Output :

```
Ram of age 42 work as Sales Man
```