Java Object Class Methods

In Java, every class directly or indirectly inherits from the `Object` class. It is the **root class** of the Java class hierarchy and is part of the `java.lang` package. Understanding the `Object` class is essential because it provides **default methods** that every Java object can use.

🔸 Why `Object` Class is Important

- All classes in Java inherit from it.
- It provides general-purpose methods like `toString()`, `equals()`, `hashCode()`, etc.
- These methods can be **overridden** to give class-specific behavior.

✅ Common Methods of `Object` Class (with Examples)

Method	Purpose
`toString()`	Returns string representation of the object
`equals(Object obj)`	Compares two objects for equality
`hashCode()`	Returns an integer hash code
`getClass()`	Returns runtime class of the object
`clone()`	Creates and returns a copy of the object
`finalize()`	Called by garbage collector before object is destroyed
`wait()`, `notify()`, `notifyAll()`	Used for thread synchronization

🔹 1. `toString()` Method

➤ Purpose:

Returns a **string** that represents the object. Default: `ClassName@HexHashCode`

```java
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // Override toString()
    public String toString() {
        return "Student{id=" + id + ", name=" + name + "}";
    }

    public static void main(String[] args) {
        Student s1 = new Student(101, "Alice");
        System.out.println(s1);  // Automatically calls s1.toString()
    }
}
```

Output:

```bash
Student{id=101, name=Alice}
```

🔹 2. `equals()` Method

➤ Purpose:

Compares two objects for **logical equality** (not reference).

```java
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // Override equals
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Student)) return false;

        Student s = (Student) o;
        return this.id == s.id && this.name.equals(s.name);
    }

    public static void main(String[] args) {
        Student s1 = new Student(101, "Alice");
        Student s2 = new Student(101, "Alice");

        System.out.println(s1 == s2);        // false (reference check)
        System.out.println(s1.equals(s2));   // true (logical check)
    }
}
```

🔹 3. `hashCode()` Method

➤ Purpose:

Returns an **integer** used in hashing-based collections like **`HashMap`**.

```java
class Student {
    int id;

    Student(int id) {
        this.id = id;
    }
    public int hashCode() {
        return id * 31;
    }

    public static void main(String[] args) {
        Student s = new Student(10);
        System.out.println(s.hashCode());  // 310
    }
}
```

💡 **Note:** If you override **`equals()`**, you should also override **`hashCode()`**.

🔹 4. `getClass()` Method

➤ Purpose:

Returns the **runtime class** of an object.

```java
class MyClass {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.getClass().getName());  // MyClass
    }
}
```

🔹 5. `clone()` Method

➤ Purpose:

Creates and returns a **copy** of an object. Must implement **`Cloneable`** interface.

```java
class Student implements Cloneable {
    int id;

    Student(int id) {
        this.id = id;
    }

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public static void main(String[] args) throws CloneNotSupportedException {
        Student s1 = new Student(100);
        Student s2 = (Student) s1.clone();

        System.out.println(s1.id);  // 100
        System.out.println(s2.id);  // 100
    }
}
```

🔹 6. `finalize()` Method

➤ Purpose:

Called by **garbage collector** before object is destroyed. Rarely used today.

```java
class MyClass {
    protected void finalize() {
        System.out.println("Object is being destroyed");
    }

    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj = null;
        System.gc();  // Request GC
    }
}
```

🔹 7. `wait()`, `notify()`, `notifyAll()` Methods

These are used in **multithreading** for inter-thread communication.

```java
class Shared {
    synchronized void print() {
        try {
            wait(); // Thread will wait until notified
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Printed after notify");
    }

    synchronized void trigger() {
        notify();  // Wakes up waiting thread
    }
}
```

✅ Summary Table

Method	Can be Overridden	Common Use
`toString()`	✅	Logging, debugging
`equals()`	✅	Comparing objects logically
`hashCode()`	✅	Storing in HashSet, HashMap
`getClass()`	❌	Reflection
`clone()`	✅	Object copy
`finalize()`	✅ (deprecated)	Cleanup before GC
`wait()/notify()`	❌	Thread sync

Would you like a small Java mini-project that shows usage of these methods together?