CLASS: Object

The `Object` class is the **supermost** class in Java. Every class in Java implicitly or explicitly extends it, making it the ultimate ancestor in the class hierarchy.

- **Direct Child Class:** A class that does **not extend** any other class explicitly.
- **Indirect Child Class:** A class that extends **any other class**, which eventually extends `Object`.

```java
class A {/* ... */}
class B extends A {/* ... */}
/*
 * A -> Direct child of Object.
 * B -> Indirect child of Object.
*/
```

Common Methods in Object Class

- `hashCode()`
- `toString()`
- `equals()`
- `finalize()`
- `clone()`
- `wait()` | `wait(long)` | `wait(long, int)`
- `notify()`
- `notifyAll()`

`hashCode()`

Generates a **unique identifier** (integer) for each object. This identifier is typically based on the object's memory address.

```java
class HashCodeExample {
    public static void main(String[] args) {
        HashCodeExample obj1 = new HashCodeExample();
        HashCodeExample obj2 = new HashCodeExample();
        System.out.println(obj1.hashCode());
        System.out.println(obj2.hashCode());
    }
}
```

- JVM uses hash codes in **hash-based data structures** like `Hashtable`, `HashMap`.
- You can override this method to customize hash code generation.

```java
class HashCode {
    int id;
    HashCode() { this.id = (int)(Math.random() * 100); }

    public int hashCode() {
        return 31 * id;
    }

    public static void main(String[] args) {
        HashCode obj1 = new HashCode();
        HashCode obj2 = new HashCode();
        System.out.println(obj1.hashCode());
        System.out.println(obj2.hashCode());
    }
}
```

`toString()`

Returns a **string representation** of the object.

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

- Overridden in classes like `String`, `StringBuffer`, `Integer`, and `ArrayList`.

```java
class ToStringExample {
    public static void main(String[] args) {
        ToStringExample ob = new ToStringExample();
        String ob1 = new String("abc");
        Integer ob2 = new Integer(12);
        ArrayList<String> ob3 = new ArrayList<>();
        ob3.add("abc"); ob3.add("def");

        System.out.println(ob);
        System.out.println(ob1);
        System.out.println(ob2);
        System.out.println(ob3);
    }
}
```

`equals()`

Used to **compare object references** by default. Can be **overridden** to compare **object contents**.

```java
class Student {
    String name;
    Student(String name) { this.name = name; }

    public boolean equals(Object o) {
        return this.name.equals(((Student)o).name);
    }

    public static void main(String[] args) {
        Student s1 = new Student("abc");
        Student s2 = new Student("abc");
        System.out.println(s1.equals(s2)); // true
    }
}
```

```java
class Student {
    int id;
    Student(int id) { this.id = id; }

    public boolean equals(Object o) {
        return this.id == ((Student)o).id;
    }

    public static void main(String[] args) {
        Student s1 = new Student(5);
        Student s2 = new Student(8);
        System.out.println(s1.equals(s2)); // false
    }
}
```

`finalize()`

The `finalize()` method is called by the Garbage Collector **before** an object is destroyed. It is used to perform cleanup operations.

Ways Objects Become Useless:

- **Nullifying reference**:

```java
Student ob = new Student(10);
ob = null;
```

- **Reassigning reference**:

```java
ob = new Student(20);
```

- **Local objects** are collected after method execution:

```java
class A {
    public static void func() {
        A a = new A();
    }

    public static void main(String[] args) {
        func();
    }
}
```

- **Cyclic references**:

```java
class Example {
    Example ref;
    public static void main(String[] args) {
        Example e1 = new Example();
        Example e2 = new Example();
        Example e3 = new Example();
        e1.ref = e2; e2.ref = e3; e3.ref = e1;
        e1 = null; e2 = null; e3 = null;
    }
}
```

Requesting JVM to Call Garbage Collector (GC)

You can request garbage collection via:

1. **Using `System` class:**

```java
System.gc();
```

2. **Using `Runtime` class:**

```java
Runtime.getRuntime().gc();
```

> Note: JVM makes the final decision to run the garbage collector.

Example:

```java
class Demo {
    public static void main(String[] args) {
        Demo d = new Demo();
        d = null;
        System.gc();
        System.out.println("End of Main");
    }

    public void finalize() {
        System.out.println("finalize method called");
    }
}
```

You can also **explicitly call** `finalize()`, but that won't destroy the object:

```java
class Demo {
    public static void main(String[] args) {
        Demo d = new Demo();
        d.finalize();
        d.finalize();

        Demo d2 = new Demo();
        d2 = null;

        System.gc();
        System.out.println("End of main");
    }

    public void finalize() {
        System.out.println("finalize method called");
    }
}
```

`clone()`

- Used to **create an exact copy** of an object.
- Requires implementing the `Cloneable` interface.

Syntax:

```java
protected Object clone() throws CloneNotSupportedException
```

```java
class Demo implements Cloneable {
    int x = 20;

    public static void main(String[] args) throws
CloneNotSupportedException {
        Demo d = new Demo();
        System.out.println(d.x); // 20
        Demo d1 = (Demo) d.clone();
        System.out.println(d1.x); // 20
    }
}
```