JAVA BASE OBJECT

(methods available in java.lang.object)

JAVA BASE OBJECT

JAVA OBJECT METHODS

Available in *java.lang.object*

- *equals*
- *hashcode*
- *toString*
- *wait*
- *notify*

METHODS TO OVERRIDE

When writing new class **always** override

- *toString*
- *equals*
- *hashcode*

STRING REPRESENTATION

- *toString* method decides the String representation of the object
- Used by *System.out.println* and loggers
- Override *toString* to provide more readable representation of the object

EXAMPLE USAGE

```
// Immutable complex number class
public final class Complex {
    private final double re;
    private final double im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    // ... code portion left out here (refer to code samples)

    @Override public String toString() {
        return "(" + re + " + " + im + "i)";
    }
}
```

OBJECT EQUALITY

- == compares references (pointers for people coming from the C world)
- For Object Equality use .*equals* instead of ==
- As always override *equals* method
- **Note** Use IDE features to generate the *equals* and *hashcode*

EXAMPLE USAGE

```
private static final Complex ORIGIN = new Complex(0, 0);

public static void main(String[] args) {
    final Complex c1 = new Complex(0, 0);
    if(isOriginRef(c1)) {
        System.out.println("Origin and C1 has the same reference");
    }
    if(isOrigin(c1)) {
        System.out.println("Point C1 is at origin");
    }
}
// Wrong way to compare if the point is origin
// may not what you have envisioned
private static boolean isOriginRef(final Complex c) {
    // Checking equality based on reference
    return c == ORIGIN;
}
// Correct way
private static boolean isOrigin(final Complex c) {
    // equality based on .equals
    return ORIGIN.equals(c);
}
```

OVERRIDE HASHCODE

- *HashMap* and *HashSet* rely on *hashcode*
- *hashcode* is used when inserting / retrieving items
- Always override *equals* when overriding *hashcode*
- Override *hashcode* (and *equals*) when using Java collections

EXAMPLE USAGE

```
// Immutable complex number class
public final class Complex {
    private final double re;
    private final double im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }
    // ... code portion left out here (refer to code samples)
    @Override public boolean equals(Object o) {
        if (o == this)
            return true;
        if (!(o instanceof Complex))
            return false;
        Complex c = (Complex) o;
        // Using compare instead of ==
        return Double.compare(c.re, re) == 0
            && Double.compare(c.im, im) == 0;
    }
    @Override public int hashCode() {
        return 31 * Double.hashCode(re) + Double.hashCode(im);
    }
}
```

SUMMARY

When creating a new Class

- Write the proper String representation using *toString*
- Better String representation makes for better debugging support
- Override *hashCode* and *equals* for using them in Java Collections
- Avoid writing *hashCode* and *equals* manually (Use IDE support to auto generate)

REFERENCES

- Java API documentation
- Effective Java
- Java Concurrency in Practice
- "Gang of Four" Design Patterns
- SOLID Principles