Overriding and Object

Override

Override	provide a new implementation for a method in the subclass.

- Overriding allows two objects related by inheritance to use the same naming convention for the methods that accomplish the same task different ways
- Static methods can be inherited, but not overridden (simple hides the superclass' method)
- When a child class defines a method with the same signature as the parent, the child's version overrides the parent.
- @Override annotation before the method signature

Don't Override Instance Variables

Child class already has the variable, which can lead to problems

Override Example

Superclass

```java
public class Person {

    protected String name;

    public Person(String n) {
        this.name = n;
    }

    public String toString() {
        return "Hello, my name is " + name;
    }
}
```

Subclass

```java
public class Student extends Person {
    protected int yr;

    public Student(String name, int year) {
        super(name);
        this.yr = year;
    }

    public Student(String name) {
        this(name, -1);
    }

    @Override
    public String toString() {
        return "Yo, my name is " + name;
    }
}
```

Object Class

Object Class	Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

- The Object class contains:
    - equals()
    - toString()
    - clone()
    - Other Methods
- Overriding equals()
    - Check if object is not null
    - Check for reference equality (==)
    - Check if the other object is *instanceof* class or Classes are equals (getClass())
        - instanceof will include subtypes
        - getClass() does not include subtypes, they have to be identical
    - Cast other object to intended class (guaranteed to work after *instanceof* check)
    - Check that each “significant” field in the other object equals(Object) the corresponding field in this object.

Example of Overriding Object Class

```java
public class GrizzlyBear {
    protected String name;

    public GrizzlyBear(String name) {
        this.name = name;
    }
}
```

equals() with *instanceof*

```java
@Override
public boolean equals(Object other) {
    if (null == other) {
        return false; }
    if (this == other) {
       return true;
    }
    if (!(other instanceof GrizzlyBear)) {
        return false;
    }
    GrizzlyBear that = (GrizzlyBear) other;
    return this.name.equals(that.name);
}
```

toString()

```java
@Override
public String toString() {
    return name;
}
```

Example of Overriding Object Class in a Subclass

```java
public class CanadianGrizzlyBear extends GrizzlyBear {
    protected String province;

    public CanadianGrizzlyBear(String name, String p) {
        super(name);
        this.province = province;
    }
}
```

equals() with getClass()

```java
@Override
public boolean equals(Object other) {
    if (null == other) {
        return false; }
    if (this == other) {
        return true;
    }
    if (getClass() != o.getClass()){
        return false;
    }
    CanadianGrizzlyBear that = (CanadianGrizzlyBear) other;
    return this.name.equals(that.name) && this.province.equals(that.province);
}
```

toString()

```java
@Override
public String toString() {
    return super.toString() + " from" + this.province + ", Canada";
}
```

Glossary

instanceof	• tests whether the object reference on the left-hand side (LHS) is an instance of the type on the right-hand side (RHS) or some subtype.
Override	provide a new implementation for a method in the subclass.
Object Class	Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.