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

```
1  public class Person {
2
3      protected String name;
4
5      public Person(String n) {
6          this.name = n;
7      }
8
9      public String toString() {
10         return "Hello, my name is " + name;
11     }
```

Subclass

```
1  public class Student extends Person {
2      protected int yr;
3
4      public Student(String name, int year) {
5          super(name);
6          this.yr = year;
7      }
8
9      public Student(String name) {
10         this(name, -1);
11     }
12
13     @Override
14     public String toString() {
15         return "Yo, my name is " + name;
16     }
```

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

```
1  public class GrizzlyBear {
2      protected String name;
3
4      public GrizzlyBear(String name) {
5          this.name = name;
6      }
```

equals() with *instanceof*

```
1    @Override
2  public boolean equals(Object other) {
3      if (null == other) {
4          return false; }
5      if (this == other) {
6         return true;
7      }
8      if (!(other instanceof GrizzlyBear)) {
9          return false;
10     }
11     GrizzlyBear that = (GrizzlyBear) other;
12     return this.name.equals(that.name);
13 }
```

toString()

```
1      @Override
2      public String toString() {
3          return name;
4      }
```

Example of Overriding Object Class in a Subclass

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

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

equals() with getClass()

```
@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()

```
@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.