# Lab 2. hashCode and equals

> Author: Yida Tao

Suppose we have an `Employee` class and its client code as follows:

```
public class Employee {

    private Long id;
    private String name;

    public Employee(Long id, String name) {
        this.name = name;
        this.id = id;
    }
}
```

```
public class EquivalenceExample {
    public static void main(String[] args) {
        Employee e1 = new Employee(1L, "John");
        Employee e2 = new Employee(1L, "John");
        Employee e3 = new Employee(2L, "Mary");

        List<Employee> employeeList = new ArrayList<>();
        employeeList.add(e1);
        employeeList.add(e2);
        employeeList.add(e3);


        Map<Employee, Integer> map = new HashMap<>();
        Integer count;
        for(Employee e : employeeList){
            if ((count = map.get(e)) == null) {
                map.put(e, 1);
            } else {
                map.put(e, 1 + count);
            }
        }

        System.out.println(map);
    }
}
```

**Q1**: what's the output when we execute the `main` method? Why?

Now, suppose we implement the `hashCode` and `equals` method as follows:

```
@Override
public int hashCode() {
    int result = id.hashCode();
    result = 31 * result + name.hashCode();
    return result;
}
```

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Employee employee = (Employee) o;

    if (!id.equals(employee.id)) return false;
    return name.equals(employee.name);

}
```

Add one of these methods or both in `Employee`, then execute the `main` method and observe the results.

**Q2**: what's the output when we override **only** the `hashCode` method and execute the `main` method? Why?

**Q3**: what's the output when we override **only** the `equals` method and execute the `main` method? Why?

**Q4**: what's the output when we override **both** the `hashCode` and `equals` method and execute the `main` method? Why?

**Hint**: check lecture notes to see how `HashMap` locates a key.

Finally, add the following code to the end of the `main` method and observe the execution result.

```
    e3.setId(3L);
    map.put(new Employee(3L, "Mary"), 3);
    System.out.println(map);
```

We'll see that the key `Employee{id=3, name='Mary'}` appears twice in the HashMap. This is because the `hashCode` return value changes since it now depends on `id`. To avoid such confusion, we probably want to make the `Employee` class immutable (e.g., data fields are `private` and `final` with no setters.)