java对象的比较

本节目标

> - Object.equals
> - Comparable<E>
> - Comparator<E>

1. 关于对象值相等的比较

1.1 == vs equals

p == q 表示的是 p 和 q 两个引用指向同一个对象

p.equals(q) 表示 p 指向的对象 和 q 指向的对象是否是值语义相等的

1.2 示例

覆写 equals 前

```
public class Card {
    public int rank;    // 数值
    public String suit; // 花色

    public Card(int rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }
}
```

```
Card p = new Card(1, "♠");
Card q = new Card(1, "♠");
Card o = p;

p == o; // true
p == q; // false

p.equals(o);    // true 因为如果不覆写 equals，默认的 equals 逻辑就是引用比较
p.equals(q);    // false    因为如果不覆写 equals，默认的 equals 逻辑就是引用比较
```

覆写 equals 后

```
public class Card {
    public int rank;    // 数值
    public String suit; // 花色

    public Card(int rank, String suit) {
```

```
        this.rank = rank;
        this.suit = suit;
    }

    @Override
    public boolean equals(Object o) {
        if (this == 0) {
            return true;
        }

        if (o == null || !(o instanceof Card)) {
            return false;
        }

        Card c = (Card)o;
        return rank == c.rank
            && suit.equals(c.suit);
    }
}
```

```
Card p = new Card(1, "♠");
Card q = new Card(1, "♠");
Card o = p;

p == o; // true
p == q; // false

p.equals(o);    // true
p.equals(q);    // true
```

注意： 一般覆写 equals 的套路就是上面演示的

1. 如果指向同一个对象，返回 true
2. 如果传入的为 null，返回 false
3. 如果传入的对象类型不是 Card，返回 false
4. 按照类的实现目标完成比较，例如这里只要花色和数值一样，就认为是相同的牌
5. 注意下调用其他引用类型的比较也需要 equals，例如这里的 suit 的比较

2. 关于对象值大于、等于、小于的比较-基于自然顺序

2.1 认识 Comparable<E>

```
public interface Comparable<E> {
    // 返回值:
    // < 0: 表示 this 指向的对象小于 o 指向的对象
    // == 0: 表示 this 指向的对象等于 o 指向的对象
    // > 0: 表示 this 指向的对象等于 o 指向的对象
    int compareTo(E o);
}
```

2.2 示例

```
public class Card implements Comparable<Card> {
    public int rank;    // 数值
    public String suit; // 花色

    public Card(int rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }

    // 根据数值比较，不管花色
    // 这里我们认为 null 是最小的
    @Override
    public int compareTo(Card o) {
        if (o == null) {
            return 1;
        }
        return rank - o.rank;
    }
}
```

```
Card p = new Card(1, "♠");
Card q = new Card(2, "♠");
Card o = new Card(1, "♠");

p.compareTo(o);     // == 0，表示牌相等
p.compareTo(q);     // < 0，表示 p 比较小
q.compareTo(p);     // > 0，表示 q 比较大
```

3. 关于对象值大于、等于、小于的比较-基于比较器

3.1 认识 Comparator<T>

```
public interface Comparator<T> {
    // 返回值:
    // < 0: 表示 o1 指向的对象小于 o2 指向的对象
    // == 0: 表示 o1 指向的对象等于 o2 指向的对象
    // > 0: 表示 o1 指向的对象等于 o2 指向的对象
    int compare(T o1, T o2);
}
```

3.2 示例

```java
public class Card implements Comparable<Card> {
    public int rank;    // 数值
    public String suit; // 花色

    public Card(int rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }
}
```

```java
public class CardComparator implements Comparator<Card> {
    // 根据数值比较，不管花色
    // 这里我们认为 null 是最小的
    @Override
    public int compare(T o1, T o2) {
        if (o1 == o2) {
            return 0;
        }

        if (o1 == null) {
            return -1;
        }

        if (o2 == null) {
            return 1;
        }

        return o1.rank - o2.rank;
    }
}
```

```java
Card p = new Card(1, "♠");
Card q = new Card(2, "♠");
Card o = new Card(1, "♠");

Comparator<Card> cmptor = new CardComparator();

cmptor.compare(p, o);       // == 0，表示牌相等
cmptor.compareTo(p, q);     // < 0，表示 p 比较小
cmptor.compareTo(q, p);     // > 0，表示 q 比较大
```

4. 比较

覆写的方法	说明
Object.equals	因为所有类都是继承自 Object 的，所以直接覆写即可，不过只能比较相等与否
Comparable.compareTo	需要手动实现接口，侵入性比较强，但一旦实现，每次用该类都有顺序，属于内部顺序
Comparator.compare	需要实现一个比较器对象，对待比较类的侵入性弱，但对算法代码实现侵入性强

5. 和 java 集合框架的配合

1. 使用 contains 类似的方法，内部基本在调用元素的 equals 方法，所以要求元素覆写过 equals 方法
2. 使用 HashMap，key 的比较内部会调用 equals 方法，所以要求元素覆写过 equals 方法
3. 使用排序相关方法，内部需要进行比较，所以或者选择实现 Comparable 或者传入一个 Comparator
4. 使用 TreeMap，key 需要进行大小比较，所以或者选择实现 Comparable 或者传入一个 Comparator
5. 其他规则以此类推

内容重点总结

- XXXX
- YYYY
- ZZZZ

课后作业

- XXXX
- YYYY
- ZZZZ