Victoria University of Wellington, New Zealand
Te Whare Wananga o te Upoko o te Ika a Maui
Aotearoa

SWEN221: Software Development

18: Object Contracts

David J. Pearce & Nicholas Cameron & James Noble & Petra Malik
Engineering and Computer Science, Victoria University

Object contracts

- All classes extend `Object`
  - Some useful methods for all objects
- Java API is general purpose
  - To take advantage, your objects must satisfy some contracts
- We'll look at some exemplars
  - But, there's more – read the docs!

Equality

- Should be easy, right?

```
class Coordinate {
  private int x, y;
  public Coordinate(int x, int y) {
     this.x = x; this.y = y;
  }
  public void main(String[] args) {
    Coordinate c1 = new Coordinate(3, 4);
    Coordinate c2 = new Coordinate(3, 4);
    System.out.println(c1 == c2);
} }
```

- What gets printed?

  A) **`true`** B) **`false`** C) Other

Equality

```
class Coordinate {
  private int x, y;
  public Coordinate(int x, int y) {
     this.x = x; this.y = y;
  }
  public boolean equals(Object o) {
    …
  }
  public void main(String[] args) {
    Coordinate c1 = new Coordinate(3, 4);
    Coordinate c2 = new Coordinate(3, 4);
    System.out.println(c1.equals(c2));
} }
```

- Why?
  - '==' gives **reference** equality
  - Use `equals(Object)` for **value equality**:

Equality

- Need to override `Object.equals()`:
  - "It shall be ***reflexive***: for any non-null reference value x, x.equals(x) should return true."
  - "It shall be ***symmetric***: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true."
  - "It shall be ***transitive***: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true."
  - "It shall be ***consistent***: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified."
  - "For any non-null reference value x, x.equals(null) should return false."

What’s wrong with this?

```
public final class InsensitiveStr {
 private String s;
 public InsenstiveStr(String x) { s=x.toLowerCase(); }
 public boolean equals(Object o) {
  if (o instanceof InsensitiveStr) {
    InsensitiveStr c =(InsensitiveStr) o;
    return s.equals(c.s);
  } else if (o instanceof String) {
    return s.equalsIgnoreCase((String) o);
  }
  return false;
}}
```

A) Not Reflexive B) Not Symmetric C) Not Transitive

What's wrong with this?

```
public class Par {
 private int data;
 public Par (int data) { this.data = data; }
 public boolean equals(Object o) {
  if(o instanceof Par) { return data==((Par)o).data; }
  else {return false; }
}}
public final class Child extends Par {
 private int data2;
 public boolean equals(Object o) {
  if (o instanceof Child) {
   return data2==((Child)o).data2 && super.equals(o);
  } else {return false; }
}}
```

A) Not Reflexive B) Not Symmetric C) Not Transitive

Fixed

```
public class Par {
 protected int data;
 public Par (int data) { this.data = data; }
 public boolean equals(Object o) {
  if(o != null && o.getClass() == this.getClass()) {
    return data==((Par)o).data;
  } else { return false; }
}}
public final class Child extends Par {
 private int data2;
 public boolean equals(Object o) {
  if (o instanceof Child) {
   return data2==((Child)o).data2 && data == o.data; }
  else {return false; }
}}
```

Object.hashCode()

- Used by `HashMap` and `HashSet` (and others)

- If override `equals`, should override `hashCode`
  - Otherwise you will get some odd bugs
  - Default `hashCode` relies on object's address

- Contract:
  - *Consistent* – shouldn't change unless equals changes
  - *Reflexive* with respect to `equals` – two equal objects must have the same hashcode
    - (May give different hashcodes for non-equal objects)

Consistent?

- Example:

```
class Coordinate {
  private int x, y;
  public boolean equals(Object o) {…}

  public int hashCode() {
     return 0;
  }
}
```

A) No B) Yes

Consistent?

```
class Coordinate {
  private int x, y;
  public boolean equals(Object o) {…}

  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + x;
    result = prime * result + y;
    return result;
  }
}
```

- Are we sure this is consistent?

Collections and Orderings

- Library class full of useful functionality
- Sort, min, max, reverse, search, copy, views…
- Many methods require a way to *order* a collection
  - Or require that the collection is already ordered

Comparable

- Implementing the `Comparable<T>` interface indicates that you can order objects

- Implement the `compareTo` method
  - Returns an int
    - `a.compareTo(b)` < 0 means a < b
    - `a.compareTo(b)` == 0 means a == b
    - `a.compareTo(b)` > 0 means a > b

- `Comparable<T>` is “generic”. More on generics later …

Comparator

```
class CoordComp implements Comparator<Coordinate>{

  public int compare(Coordinate a, Coordinate b) {
    return (a.x + a.y) - (b.x + b.y);
  }

}
```

- An **interface** implemented by another class
- **Generic parameter** is the class to compare
- Implement `compare` and `equals` methods
  - Same contracts as before