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

SWEN221 Software Development

Object Contracts

Thomas Kuehne

Victoria University

(slides modified from slides by
David J. Pearce & Nicholas Cameron & James Noble & Petra Malik)

Object contracts

- All classes extend `Object`
  - not only a useful “top” type,
  - also a common provider of functionality
- Object functionality typically requires adaptation
  - need to follow contracts to comply with expectations
- Only a subset of aspects covered here
  - read the Javadoc documentation for more

Equality

```
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.equals(c2));
  }
}
```

- What is printed?

A) `true`   B) `false`

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

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

Fix Attempt

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

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

Fix

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

Object.hashCode()

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

- If one overrides `equals`,
  one should override `hashCode`
  - otherwise one will get inconsistent behaviour
  - default `hashCode` relies on object's address

- Contract for `hashCode`:
  - *Consistent* – shouldn't change unless state changes
  - *Consistent* with respect to `equals` – two equal objects must have the same hashcode
    - (non-equal objects still may yield the same hashcodes)

Consistent?

- Example:

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

  public int hashCode() {
    return 1;
  }
}
```

- Is this consistent?

A) No B) Yes

Further Reading …

- http://www.angelikalanger.com/Articles/JavaSolutions/SecretsOfEquals/Equals.html