Principles of Software Construction: Objects, Design, and Concurrency

Inheritance (continued), type-checking, and behavioral contracts -- Extra slides

Spring 2014

Charlie Garrod Christian Kästner

School of Computer Science
institute for SOFTWARE RESEARCH

The `java.lang.Object`

- All Java objects inherit from `java.lang.Object`
- Commonly-used/overridden public methods:

```
String    toString()
boolean   equals(Object obj)
int       hashCode()
Object    clone()
```

Overriding `java.lang.Object`'s `.equals`

- The default `.equals`:

```
public class Object {
    public boolean equals(Object obj) {
        return this == obj;
    }
}
```

- An aside: Do you like:

```
public class CheckingAccountImpl
        implements CheckingAccount {
    @Override
    public boolean equals(Object obj) {
        return false;
    }
}
```

Recall the `.equals(Object obj)` contract

- An equivalence relation
  - Reflexive: ∀`x` `x.equals(x)`
  - Symmetric: ∀`x,y` `x.equals(y)` if and only if `y.equals(x)`
  - Transitive: ∀`x,y,z` `x.equals(y)` and `y.equals(z)` implies `x.equals(z)`
- Consistent
  - Invoking `x.equals(y)` repeatedly returns the same value unless `x` or `y` is modified
- `x.equals(null)` is always false
- `.equals()` always terminates and is side-effect free

The `.hashCode()` contract

- Consistent
  - Invoking `x.hashCode()` repeatedly returns same value unless `x` is modified
- `x.equals(y)` implies `x.hashCode() == y.hashCode()`
  - The reverse implication is not necessarily true:
    - `x.hashCode() == y.hashCode()` does not imply `x.equals(y)`
- Advice: Override `.equals()` if and only if you override `.hashCode()`

The `.clone()` contract

- Returns a *deep copy* of an object
- Generally (but not required!):
  - `x.clone() != x`
  - `x.clone().equals(x)`

A lesson in equality

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

Recall: The `java.lang.Object`

- All Java objects inherit from `java.lang.Object`
- Commonly-used/overridden public methods:
  - `String  toString()`
  - `boolean  equals(Object obj)`
  - `int  hashCode()`
  - `Object  clone()`

Complete to support equality-checking for the `Point` class.

A tempting but incorrect solution

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

```
public boolean equals(Point p) {
  return x == p.x && y == p.y;
}
```

Types must match

Recall: The `java.lang.Object`

- All Java objects inherit from `java.lang.Object`
- Commonly-used/overridden public methods:
  - `String toString()`
  - `boolean equals(Object obj)`
  - `int hashCode()`
  - `Object clone()`

`boolean equals(Point p)` does not override `boolean equals(Object obj)`

A correct solution

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
  }

  public int hashCode() {
    return 31*x + y;
  }
}
```

The `.equals(Object obj)` contract

- An equivalence relation
  - Reflexive: ∀x `x.equals(x)`
  - Symmetric: ∀x,y `x.equals(y)` if and only if `y.equals(x)`
  - Transitive: ∀x,y,z `x.equals(y)` and `y.equals(z)` implies `x.equals(z)`
- Consistent
  - Invoking `x.equals(y)` repeatedly returns the same value unless `x` or `y` is modified
- `x.equals(null)` is always false

15-214 Garrod 8

A new challenge

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
}
```

```
public class ColorPoint
      extends Point {
  private final Color color;

  public ColorPoint(int x,
                    int y,
                    Color color) {
    super(x, y);
    this.color = color;
  }
}
```

Implement `.equals` for the `ColorPoint` class.
You may assume `Color` correctly implements `.equals`

A tempting solution

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
}
```

```
public class ColorPoint
      extends Point {
  private final Color color;

  public ColorPoint(int x,
                    int y,
                    Color color) {
    super(x, y);
    this.color = color;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof ColorPoint))
       return false;
    ColorPoint cp = (ColorPoint) obj;
    return super.equals(cp) &&
           color.equals(cp.color);
}
```

A tempting solution

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
}
```

```
public class ColorPoint
      extends Point {
  private final Color color;

  public ColorPoint(int x,
                    int y,
                    Color color) {
    super(x, y);
    this.color = color;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof ColorPoint))
       return false;
    ColorPoint cp = (ColorPoint) obj;
    return super.equals(cp) &&
           color.equals(cp.color);
}
```

A problem: `p.equals(cp)` but `!cp.equals(p)`:

```
Point p = new Point(2, 42);
ColorPoint cp = new ColorPoint(2, 42, Color.BLUE);
```

More problems

```
public class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
}
```

```
public class ColorPoint
      extends Point {
  private final Color color;

  public ColorPoint(int x,
                    int y,
                    Color color) {
    super(x, y);
    this.color = color;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof Point))
       return false;
    if (!(obj instanceof ColorPoint))
       return super.equals(obj);
    ColorPoint cp = (ColorPoint) obj;
    return super.equals(cp) &&
           color.equals(cp.color);
}
```

Consider:

```
Point p = new Point(2, 42);
ColorPoint cp1 = new ColorPoint(2, 42, Color.BLUE);
ColorPoint cp2 = new ColorPoint(2, 42, Color.MAUVE);
```

An abstract solution

```
public abstract class Point {
  private final int x;
  private final int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object obj
    if (!(obj instanceof Point)
      return false;
    Point p = (Point) obj;
    return x == p.x && y == p.y;
}
```

```
public class ColorPoint
      extends Point {
  private final Color color;

  public ColorPoint(int x,
                    int y,
                    Color color) {
    super(x, y);
    this.color = color;
  }

  public boolean equals(Object obj) {
    if (!(obj instanceof ColorPoint))
       return false;
    ColorPoint cp = (ColorPoint) obj;
    return super.equals(cp) &&
           color.equals(cp.color);
}
```

```
public class PointImpl extends Point {
  public PointImpl(int x, int y) { super(x,y); }
  public boolean equals(Object obj) {
    if (!(obj instanceof PointImpl))
      return false;
    return super.equals(obj);
  }
```

The lesson

- Conforming to behavioral contracts can be difficult
- Advice:
  - Don't allow equality between distinct types
  - Be careful when inheriting from a concrete class

"Overriding the `equals` *method seems simple, but there are many ways to get it wrong and the consequences can be dire."* -- Josh Bloch

The lesson

- Conforming to behavioral contracts can be difficult
- Advice:
  - Don't allow equality between distinct types
  - Be careful when inheriting from a concrete class
- Symmetry kills:

```
public class EvilButTrue {
  public boolean equals(Object obj) {
      return obj != null;
  }
  public int hashCode() {
      return 0;
  }
}
```

"Overriding the `equals` *method seems simple, but there are many ways to get it wrong and the consequences can be dire."* -- Josh Bloch