CSC 143

Overriding methods from the Object class: equals, toString

Object class

- The ultimate superclass
  - Any object is an Object

```
 MyClass c = new MyClass();
System.out.print(c instanceof Object);
// always prints true
```

- All methods from Object are inherited by any other class
  - Should they be overridden?
  - Take a look at toString and equals.

public String toString() (1)

```
public class Car {
  private String make;
  private double weight;
  public Car(String theMake, double theWeight) {
         make = theMake;
         weight = theWeight;
  }
}
```

-
  ```
  Car c = new Car(“Ford”, 2000);
  System.out.println(c.toString());
  // prints Car@82ba41
  // name of the class + some hash code
  ```
- Override toString in Car to make it more meaningful

public String toString() (2)

```
public class Car {
  private String make;
  private double weight;
  public Car(String theMake, double theWeight) {
    make = theMake;
    weight = theWeight;
  }
  public String toString() {
    return “make = “ + make + “, weight = “ + weight;
  }
}
```

-
  ```
  Car c = new Car(“Ford”, 2000);
  System.out.println(c);
  // which is equivalent to
  System.out.println(c.toString());
  // prints make = Ford, weight = 2000.0
  ```
- Always override toString()

public boolean equals(Object o)

- Implemented in Object as

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

- OK?

```
Car c1 = new Car(“Ford”, 2000);
Car c2 = new Car(“Ford”, 2000);
System.out.println(c1.equals(c2)); // prints false
```

- Fix: override equals within Car

equals (2)

```
public class Car {
  // previous code omitted
  public boolean equals(Object o) {
     if (o instanceof Car) {
        Car c = (Car) o;
        return (this.weight == c.weight &&
                this.make.equals(c.make));
     }
     else {
        return false;
     }
  }
}
```

- o must be a Car: check with instanceof
- Use equals to compare fields that have a reference type (such as String).

Does it work?

-
  ```
  Car c1 = new Car(“Ford”, 2000);
  Car c2 = new Car(“Ford”, 2000);
  System.out.println(c1.equals(c2));
  // prints true.
  ```
- But wait!
- What if Car is inherited?

equals and inheritance

```
public class FancyCar extends Car{
      private double topSpeed;
      public FancyCar(String theMake,
                      double theWeight, double theTopSpeed){
          super(theMake, theWeight);
          topSpeed = theTopSpeed;
      }
      public boolean equals(Object o) {
          if (o instanceof FancyCar) {
             FancyCar fc = (FancyCar) o;
             return (super.equals(o) && this.topSpeed == o.topSpeed);
          }
          else {
             return false;
          }
      }
   }
```

-
  ```
  Car c = new Car(“Ford”, 2000);
  FancyCar fc = new FancyCar(“Ford”, 2000, 200);
  System.out.print(c.equals(fc)); // prints true
  System.out.print(fc.equals(c)); // prints false
  ```

What is going on?

- A FancyCar is a Car
`fc instanceof Car is true`

- A Car is not a FancyCar
`c instanceof FancyCar is false`

- One requirement of equals is that
if x.equals(y) is true, then y.equals(x) is also true.

- instanceof checks an is_a relationship

- A necessary condition for two variables to be equal is that they have the same dynamic type.

- Get the dynamic type with getClass(). Don't use instanceof.

A better equal

- In Car

```
public boolean equals(Object o) {
    if (o != null && o.getClass() == this.getClass()) {
       Car c = (Car) o;
       return (this.weight == c.weight &&
               this.make.equals(c.make));
    }
    else {return false;}
 }
```

- In FancyCar

```
public boolean equals(Object o) {
   if (o != null && o.getClass() == this.getClass()) {
      FancyCar fc = (FancyCar) o;
      return (super.equals(o) && topSpeed==fc.topSpeed);
   }
   else {return false;}
 }
```

One last word

- Check the class website for the complete code of the previous examples
- If equals is overridden, override hashcode as well. See later when talking about hash maps…