Object
Ch 7.2-7.4
Topics

1) What does Object do for us?
   a) To a string
   b) Checking equality
   c) Hash-codes
Object's Methods

- The class Object is the base class of all classes.
- Important Methods:
  - toString()
  - equals()
  - hashCode()
  - clone()
    (Copies the object; not discussed here)
toString()

• Automatically called when:
  – you concatenate an object with a string.
  – you print an object with print() or println().

• Default implementation
  – Object.toString() prints class name and hash code.
    ToStringExample@6876fb1b

• Reasonable toString() Structure
  @Override
  public String toString() {
    return getClass().getName() + "[
    + "...
    + "}"
  }

  getClass() gets the..
class Employee {
    private String name;
    private int salary;
    @Override
    public String toString() {
        return getClass().getName()
            + "[name=" + name
            + ",salary=" + salary
            + "]";
    }
}

class Manager extends Employee {
    private int bonus;
    @Override
    public String toString() {
        return super.toString()
            + "[bonus=" + bonus
            + "]";
    }
}

public static void main(String[] args) {
    Employee bob = new Employee(...);
    System.out.println(bob);
    Manager sarah = new Manager(...);
    System.out.println(sarah);
}

Employee[name=Bob,salary=50000]
Manager[name=Sarah,salary=100][bonus=200000]
equals()

• equals() checks if..

• Common error:
  Car a, b;
  if (a == b) {
    ...
  }

• Should use:
  if (a.equals(b)) {
    ...do something
  }

But Object Provides:
public boolean equals(Object obj) {
  ..
}

Override equals() only when necessary
Equality

• Each class defines equals() as needed
  – Used by many classes to check equality.
    Ex: set checking for unique items.

• Example: Strings
  – Two Strings equal if..

• Example: ICBC Database
  – Two Cars are equal if..
    even if different colours (likely one repainted)

• Override equals() and check fields
  – For objects..
  – For primitives..
Casting Safety

class Car {
  @Override
  public boolean equals(Object otherObject) {
    if (otherObject.getClass() == this.getClass()) {
      Car other = (Car) otherObject;// ... then use it.
    }
  }
}

To use otherObject, must cast to Car. However, this can throw an exception.

Check otherObject is..
(Must work for derived objects)

if (otherObject.getClass() == this.getClass()) {
  Car other = (Car) otherObject;
  // ... then use it.
}

Complete equals()

public class Car {
    private String make;
    private Date year;
    private double weight;

    @Override
    public boolean equals(Object o) {
        if (o == null) {
            return false;
        }
        if (o.getClass() != this.getClass()) {
            return false;
        }
        Car oCar = (Car) o;

        // Assumes make and year are not null!
        return (this.make.equals(oCar.make)
                && this.year.equals(oCar.year)
                && this.weight == oCar.weight);
    }
}

• Check other is null
• Check if they are different classes.
• Check if all fields the same.
public class Truck {
    private String make;
    private Date year;
    private double weight;

    public boolean equals(Truck o) {
        if (this == null) {
            return false;
        }
        if (o.getClass() != this.getClass()) {
            return false;
        }
        return (this.make.endsWith(o.make)
                && year == o.year
                && this.weight == o.weight);
    }
}
**instanceof aside**

- Some suggest using instanceof instead of .class()

```java
@Override
public boolean equals(Object otherObject) {
    if (otherObject == null) {
        return false;
    }
    if (!other instanceof Car) {
        return false;
    }
    Car other = (Car) otherObject;
    // Assumes make and year are not null!
    return (this.make.equals(other.make) && this.year.equals(other.year) && this.weight == other.weight);
}
```

- Checking identical class violates LSP: Cannot have a derived class.
- But, instanceof can hit symmetry issues: ```car.equals(sportsCar) vs sportsCar.equals(car)``` 
- equals() is hard with inheritance!
hashCode()

- A hash code is..
  - Used in hash-maps/hash-sets.
  - Object's hashCode() hashes the memory address.

![Diagram](https://www.geeksforgeeks.org/internal-working-of-hashmap-java/)
Full Example

```java
public class Car {
    private String make;
    private Date year;
    private double weight;

    @Override
    public boolean equals(Object otherObject) {
        // ... some code omitted...
        return (this.make.equals(other.make)
                && this.year.equals(other.year)
                && this.weight == other.weight);
    }

    @Override
    public int hashCode() {
        int result = make.hashCode();
        result = result * 31 + year.hashCode();
        result = result * 31 + Double().hashCode(weight);
        return result;
    }
}
```

- Sum `hashCode()`’s of all fields used in `equals()`
  - if field x an object: ..
  - if a primitive: ..
  - if an array: each element as a field.

- Multiply previous sum by 31
  - 31 is.. , odd, and efficient on some HW
hashCode() and equals()

- hashCode() and equals() should...
  - Hash tables use both methods to find elements:
    Use hashCode() to find the “bin”
    Use equals() to find the object inside the “bin”.

- ..
  - Otherwise collections may not work correctly with the class!
  - Use the same set of fields for computing equals() as for hashCode().
What is wrong with the following?

```java
@Override
public int hashCode() {
    return 42;
}
```

```java
public class Bucket {
    private String label;
    private double cost;

    public boolean equals(Object o) {
        // ... some code omitted...
        return (label.equals(o.make));
    }

    public int hashCode() {
        return Double.hashCode(cost) * 31 + label.hashCode();
    }
}
```

Works! But terrible efficiency!

```
 Depending on field not used in equals!

 x and y might show as equal(); however have different hashCode() values!
```
Summary

- Subtleties with Object:
  - `toString()`: call `super.toString()` and structure it to make sense.
  - `equals()`: check for null, class type, `super.equals()`, and all fields.
  - `hashCode()`: overload with `equals()`.