

Comparing Objects

- `boolean result = obj1.equals(obj2);`
- `int diff = obj1.compareTo(obj2);`
- `int diff = c.compare(obj1, obj2);`
 - Used in **Collections**

obj1.equals(obj2)

- The boolean method **equals** comes from the class **Object**:

```
public boolean equals(Object other)
{ ... }
```

- **Object's equals** is not very useful: compares addresses of objects
- Programmers often override **equals** in their classes

obj1.equals(obj2) (cont)

```
public class Coordinate {
    private int row, col;
    ...

    public boolean equals(Object other) {
        if (other != null &&
            other instanceof Coordinate &&
            row == ((Coordinate)other).row &&
            col == ((Coordinate)other).col )
            return true;

        return false;
    }
}
```

obj1.equals(obj2)


- **equals** is called polymorphically from library methods, such as **ArrayList's contains** or **indexOf** - that is why we have to properly override **Object's equals**.
- The **equals** method is properly defined in **String, Integer, Double**, etc.

obj1.compareTo(obj2)

- `compareTo` is an abstract method defined in the `java.util.Comparable<T>` interface:

```
public int compareTo (T other);
```

T is the data
type parameter



- Returns a positive integer if **this** is “greater than” **other**, a negative integer if **this** is “less than” **other**, zero if they are “equal.”

Sort of like **(this - other)**

obj1.compareTo(obj2) (cont)

```
public class Element implements Comparable<Element> {  
  
    private double mass; // atomic mass  
    ...  
  
    public int compareTo(Element other) {  
        // accurate to 3 decimal points  
        int iMass = (int)(mass * 1000);  
        int iOther = (int)(other.mass * 1000);  
        return iMass - iOther;  
    }  
  
    public boolean equals(Object other) {  
        return other instanceof Element &&  
            compareTo((Element)other) == 0;  
    }  
}
```

equals is not required by **Comparable**, but it is a good idea to provide it and make it agree with **compareTo**

obj1.compareTo(obj2) (cont)


- **compareTo** is called polymorphically from library methods.
- Objects of classes that implement **Comparable** are called “comparable”.
- **Strings, Integers, Doubles** are comparable.

compare(obj1, obj2)

- **compare** is an abstract method defined in the `java.util.Comparator<T>` interface:

```
public int compare (T obj1, T obj2);
```

T is the data
type parameter

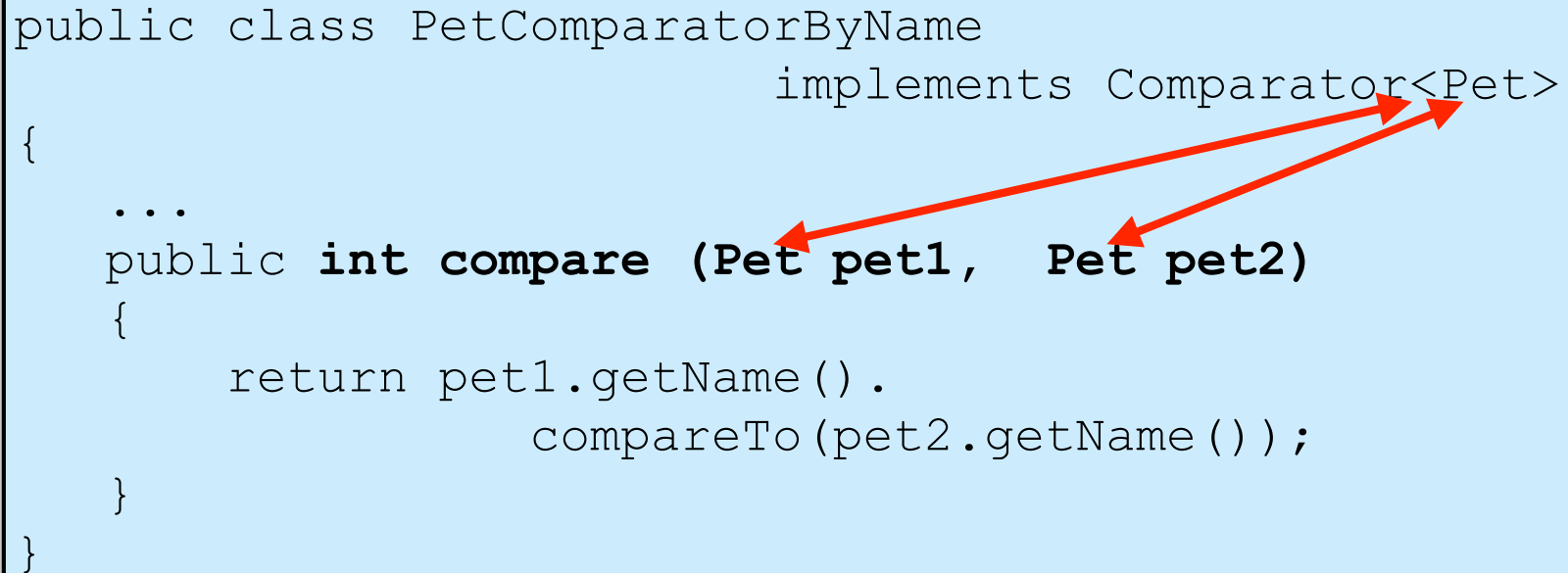


- Returns a positive integer if **obj1** is “greater than” **obj2**, a negative integer if **obj1** is “less than” **obj2**, zero if they are “equal.”

Sort of like (**obj1 - obj2**)

compare(obj1, obj2) (cont)

```
public class PetComparatorByName
    implements Comparator<Pet>
{
    ...
    public int compare (Pet pet1, Pet pet2)
    {
        return pet1.getName().
            compareTo (pet2.getName());
    }
}
```

The diagram shows two red arrows originating from the `Comparator<Pet>` interface in the code above. One arrow points to the `Pet pet1` parameter of the `compare` method, and the other points to the `Pet pet2` parameter. This illustrates that the `compare` method is implementing the `compare` method defined in the `Comparator` interface.

- A programmer can define different comparators to be used in different situations.
- **compare** is called from library methods such as `Arrays.sort(T[] arr, Comparator<T> c)` (or from your own methods that take a comparator object as a parameter)