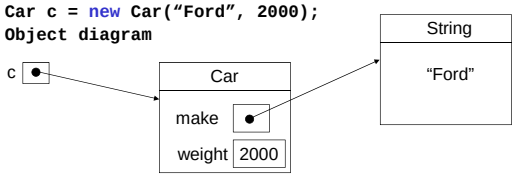
CSC 143

Making a copy of an object: shallow and deep copy

Example

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

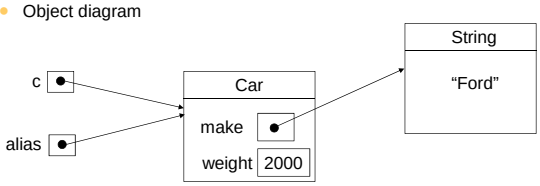
- `Car c = new Car("Ford", 2000);`
- `Object diagram`



- `How to make a copy of c?`

An alias

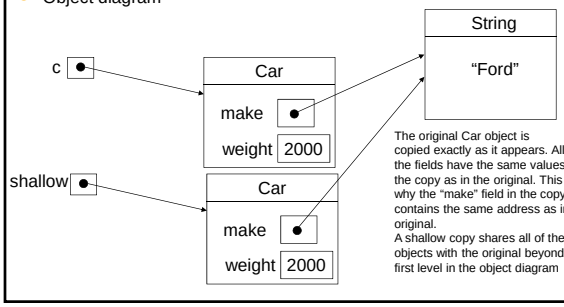
- `Car c = new Car("Ford", 2000);`
- `Car alias = c;`
- Object diagram



- Not really a copy!

A shallow copy

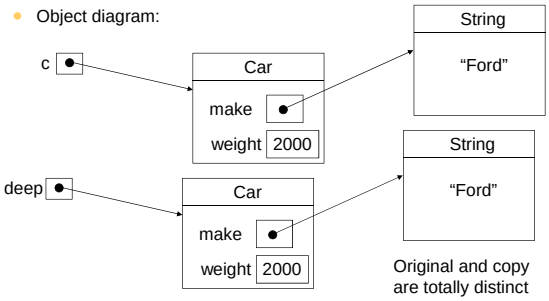
- `Car c = new Car("Ford", 2000);`
- `Car shallow = ??; // See code later`
- Object diagram



The original Car object is copied exactly as it appears. All of the fields have the same values in the copy as in the original. This is why the "make" field in the copy contains the same address as in the original.
A shallow copy shares all of the objects with the original beyond the first level in the object diagram

A deep copy

- `Car c = new Car("Ford", 2000);`
- `Car deep = ??; // See code later`
- Object diagram:



Coding

```
public class Car {
  private String make;
  private double weight;
  // many ways
  // a copy constructor
```

```
public Car(Car c) { //shallow
  this.weight = c.weight;
  this.make = c.make;
}
```

```
public Car(Car c) { // deep
  this.weight = c.weight;
  this.make = new String(c.make);
}
```

```
  // an instance method
```

```
public Car shallowCopy() {
  return new Car(this.weight,
  this.make);
}
```

```
public Car deepCopy() {
  return new Car(this.weight,
  new String(this.make));
}
```

```
  // a factory method (static method)
```

```
public static Car shallowCopy(Car c) {
  return new Car(c.weight, c.make);
}
```

```
public static Car deepCopy(Car c) {
  return new Car(c.weight, new
String(c.make));
}
```

```
}
```

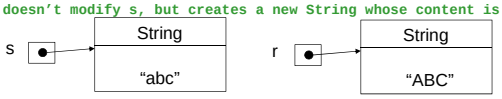
Java also offers the clone method (but clunky!)

How to choose?

- A deep copy requires more execution time and memory
- Always return a deep copy if you want to be sure that the original data can't be changed by a caller.
- Sometimes a shallow copy has the same effect as a deep copy since some Java classes are immutable.

Immutable classes

- Immutable class: a class whose instances CANNOT be modified once created.
- The class doesn't offer any method that would allow a client to modify an instance of the class.
- The String class is immutable
-
```
String s = new String("abc");
String r = s.toUpperCase();
// doesn't modify s, but creates a new String whose content is "ABC"
```



- Color, wrapper classes (Double, Character, etc.), File, Font are other examples of immutable classes (there are more). And of course you can write your own immutable classes.
- In our Car example, the shallow copy behaves like a deep copy (meaning that the original can't be modified even when shallow copied).