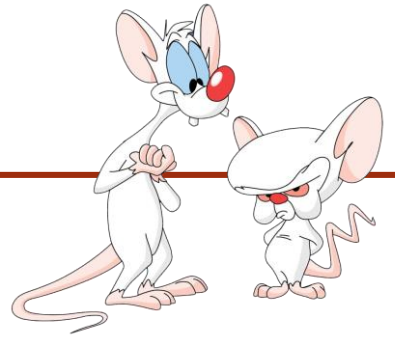
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 4-5 : OOP5 the class Object

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



OOD5

- **The** `Object` **class**

Object CLASS

THE `Object` CLASS

- `Object` is the only class in java without a superclass. All other classes have one and only one direct superclass.
- In the absence of any other specific superclass, every class is implicitly a subclass of `Object`.

Class Object

java.lang.Object

`public class` **`Object`**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

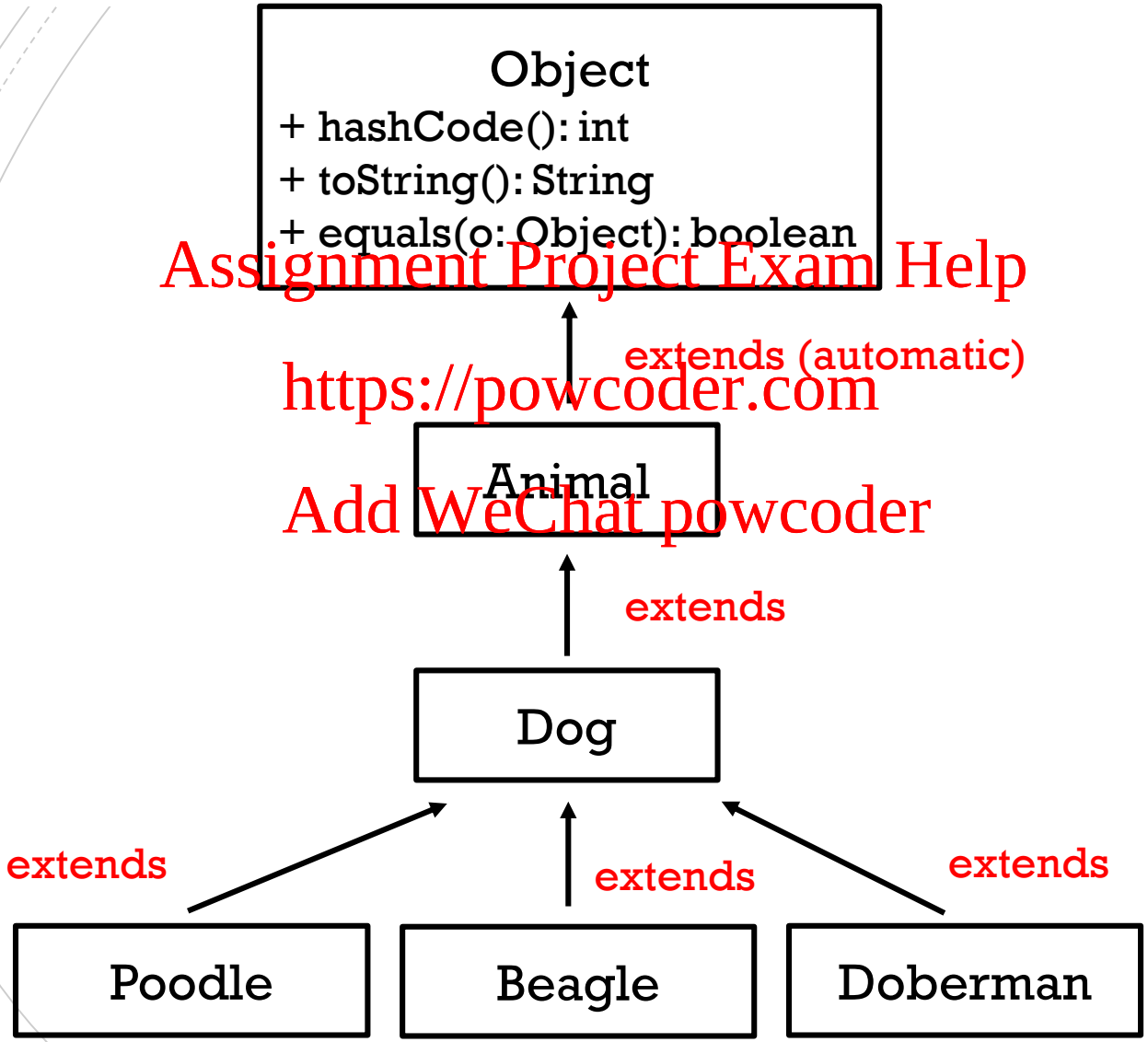
METHODS FROM `Object`

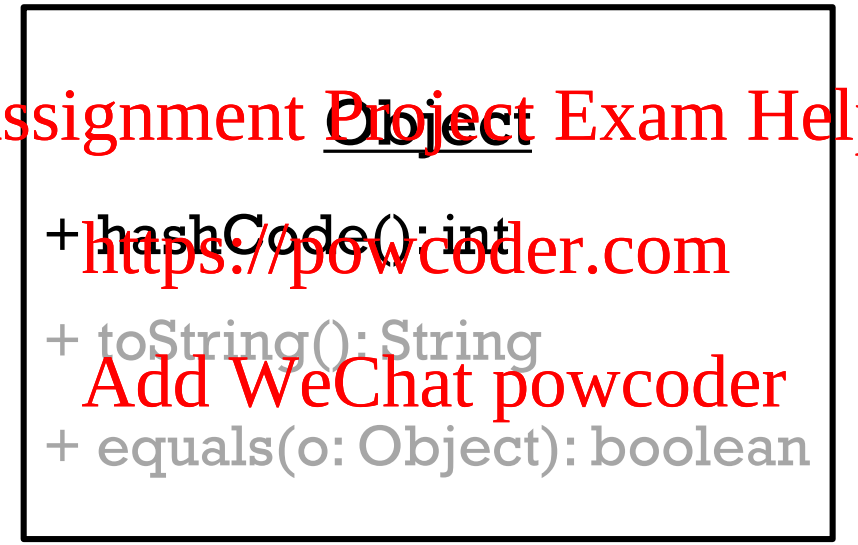
Here are some of the methods from the `Object` class:

protected **Object**	**clone**() Creates and returns a copy of this object.
boolean	**equals**(**Object** obj) Indicates whether some other object is "equal to" this one.
protected void	**finalize**() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	**getClass**() Returns the runtime class of this `Object`.
int	**hashCode**() Returns a hash code value for the object.
String	**toString**() Returns a string representation of the object.

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

HIERARCHY FOR OUR EXAMPLES





hashCode() - RETURN VALUE

- It returns a 32 bit integer associated to this object.
- "typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language".
- Use of hashCode() method : Returns a hash value that is used to search object in a collection.

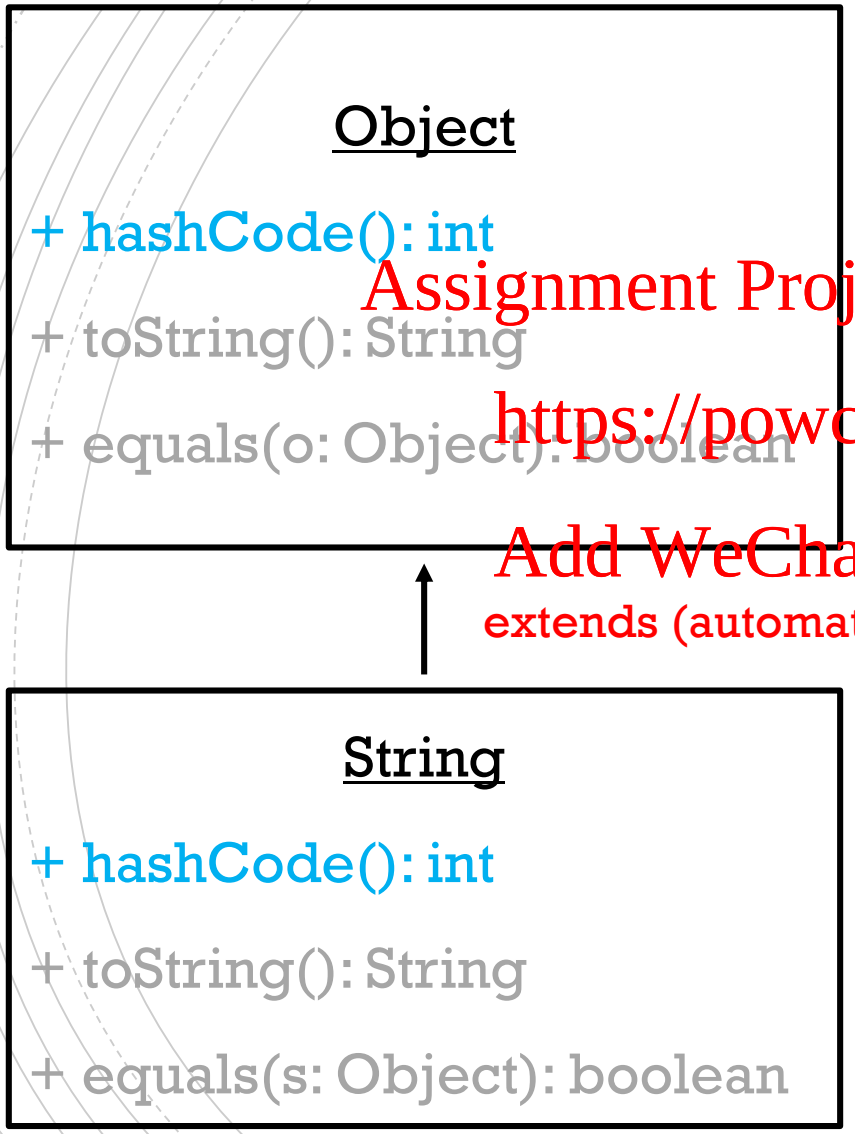
`hashCode()` - REQUIREMENTS

- "Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer."
- If `o1.equals(o2)` is `true`, then `o1.hashCode()==o2.hashCode()` should also be `true`.

Note that the converse does not need to hold!

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--

EXAMPLE



The class `String`
overrides `hashCode()`

EXAMPLE

The method `hashCode()` **from the class** `String`

hashCode

```
public int hashCode()
```


Returns a hash code for this string. The hash code for a `String` object is computed as


```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```


using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

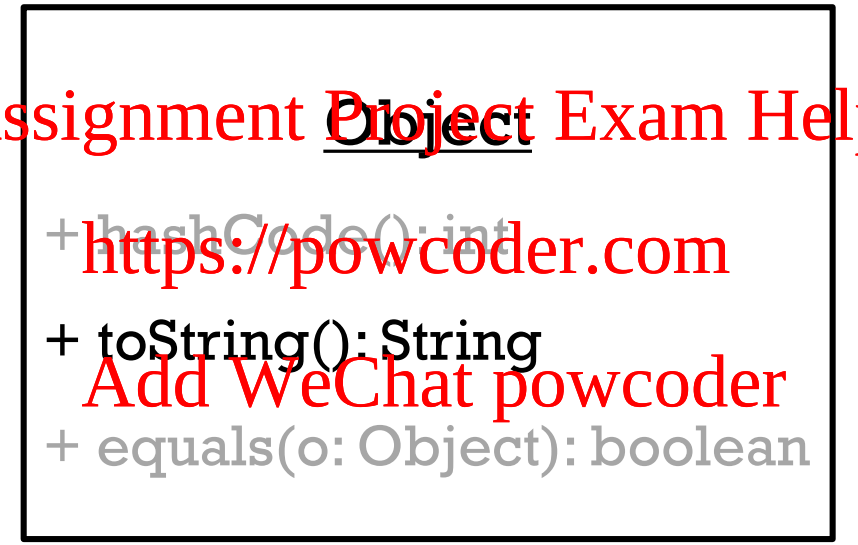
Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this object.

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html



toString()

- Returns a string representation of the object.

- It is recommended that all subclasses override this method.

- The `toString()` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character '@', and the unsigned hexadecimal representation of the hash code of the object.

EXAMPLE

```
System.out.println( new Object() );
```

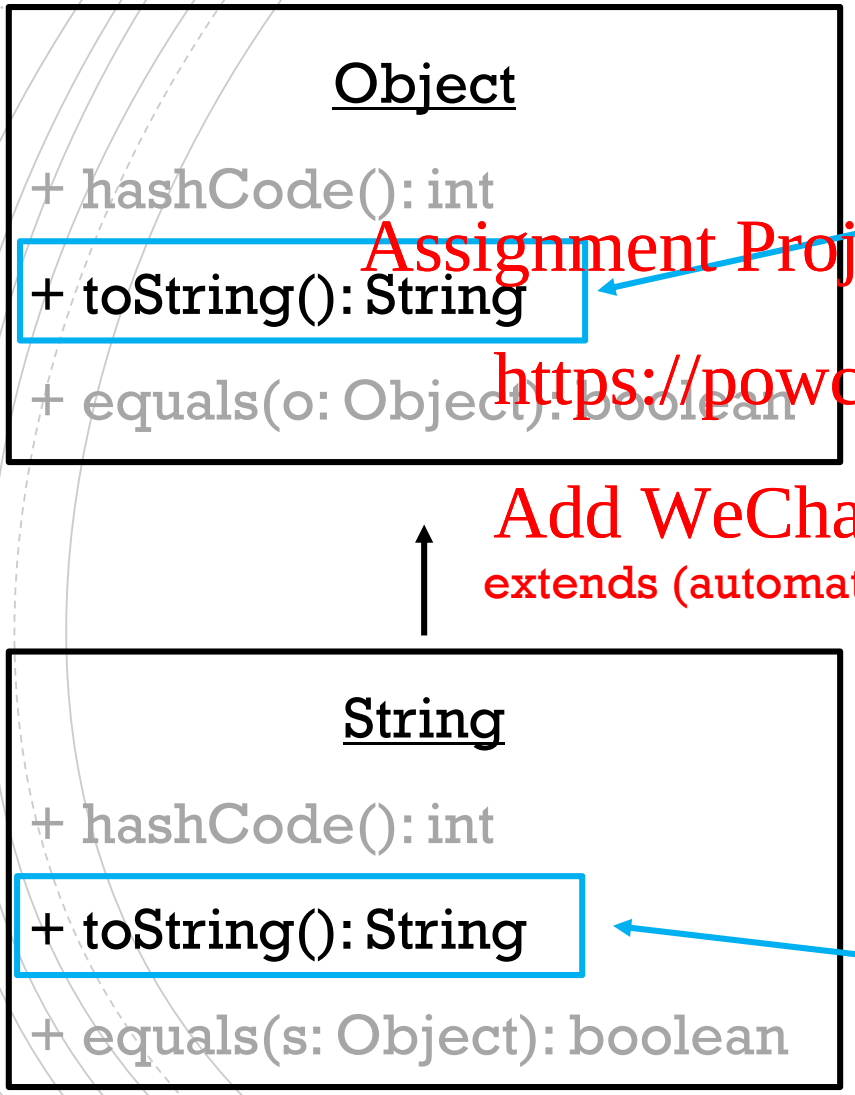

What does this print?

```
java.lang.Object@7852e922
```

package + class name

32 bit integer represented in hexadecimal

EXAMPLE



Returns the following:

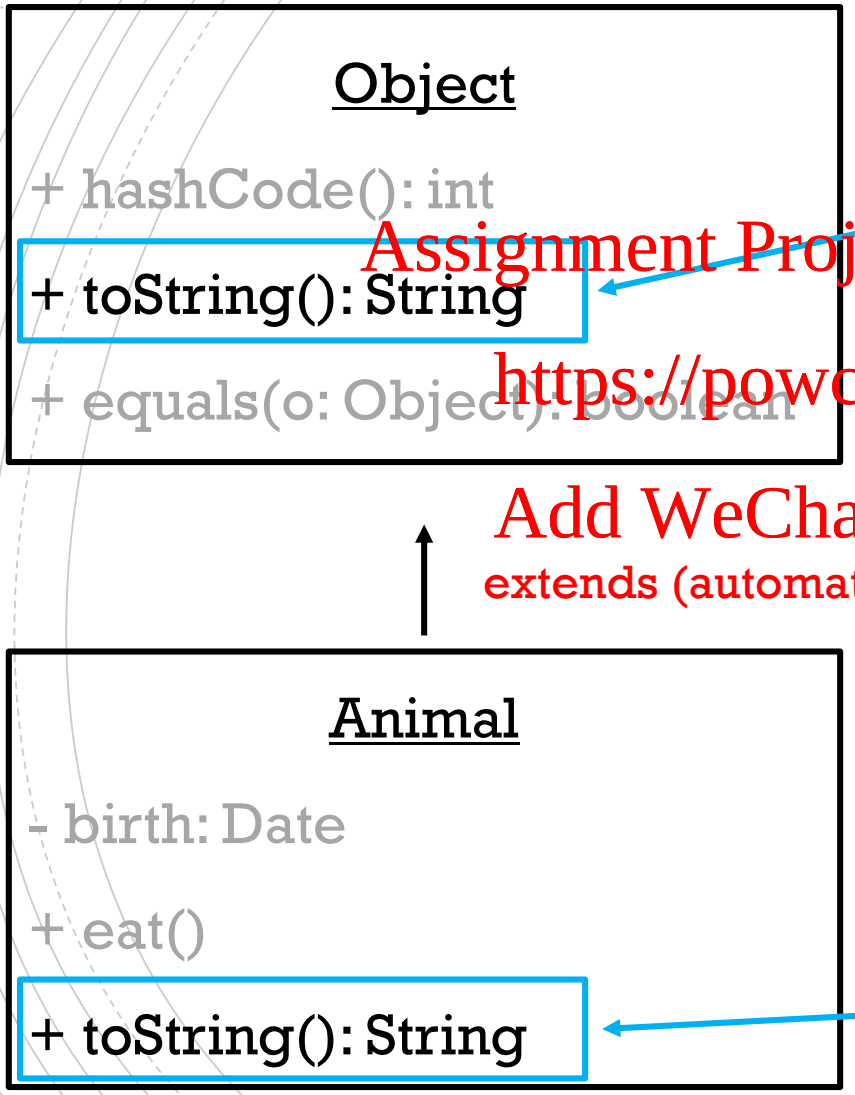
```
getClassName + "@" +
Integer.toHexString(hashCode())
```


extends (automatic)

`toString()` **is overridden in the class** `String`

Returns the object itself

EXAMPLE



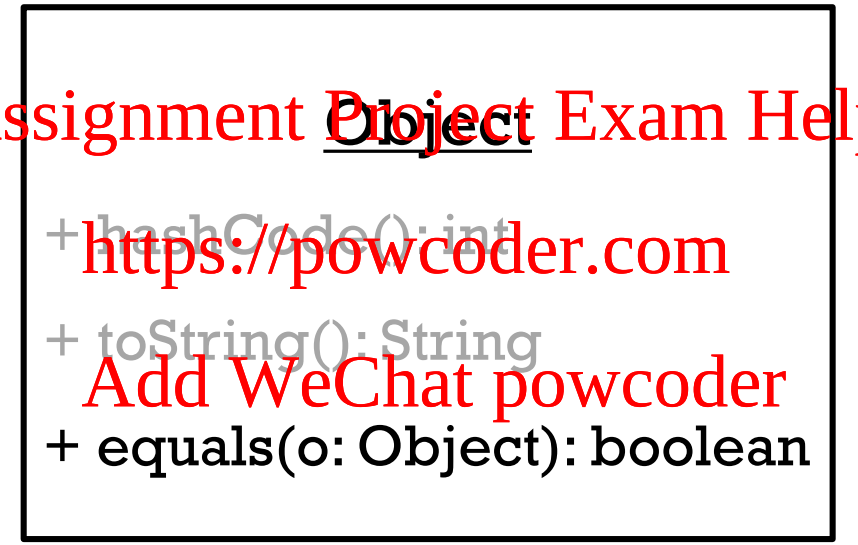
Returns the following:

```
getClassName + "@" +
Integer.toHexString(hashCode())
```


extends (automatic)

`toString()` **is overridden in the class** `Animal`

Returns… depends on your implementation!



https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

equals()

equals

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

`equals()` – IMPLEMENTATION

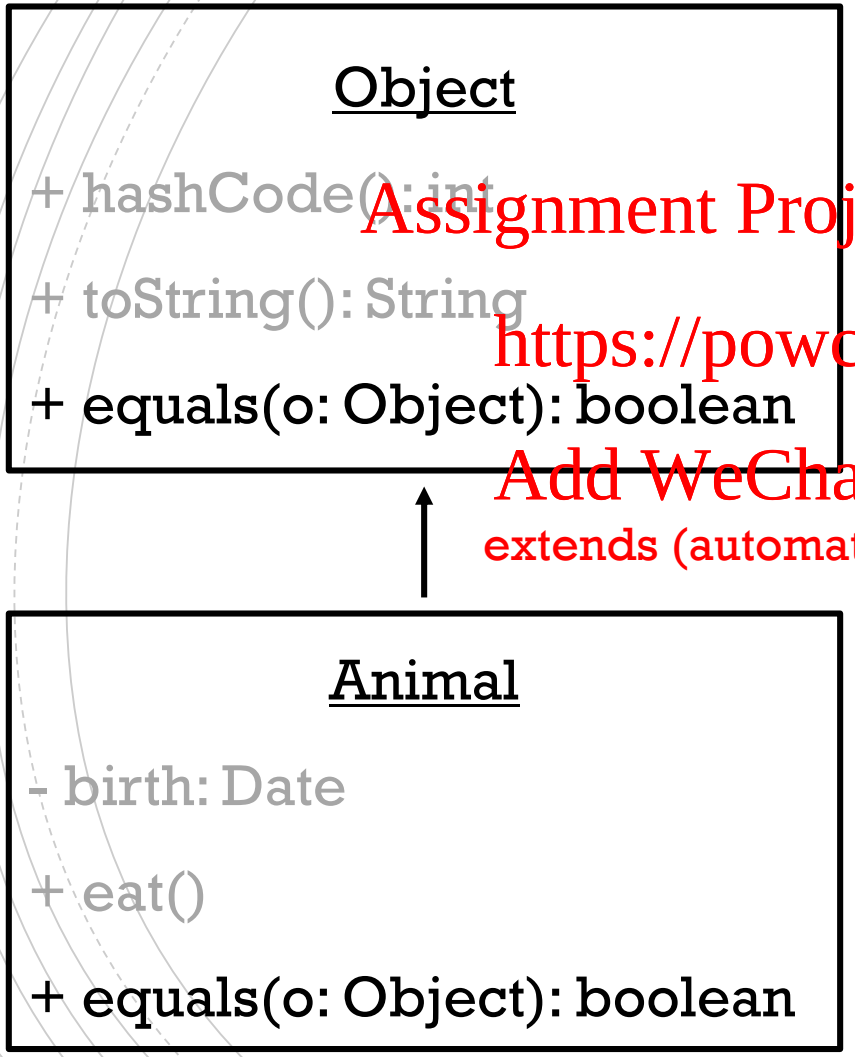
For any non-null reference values `obj1` and `obj2`,

`obj1.equals(obj2)` **returns** `true`

if and only if

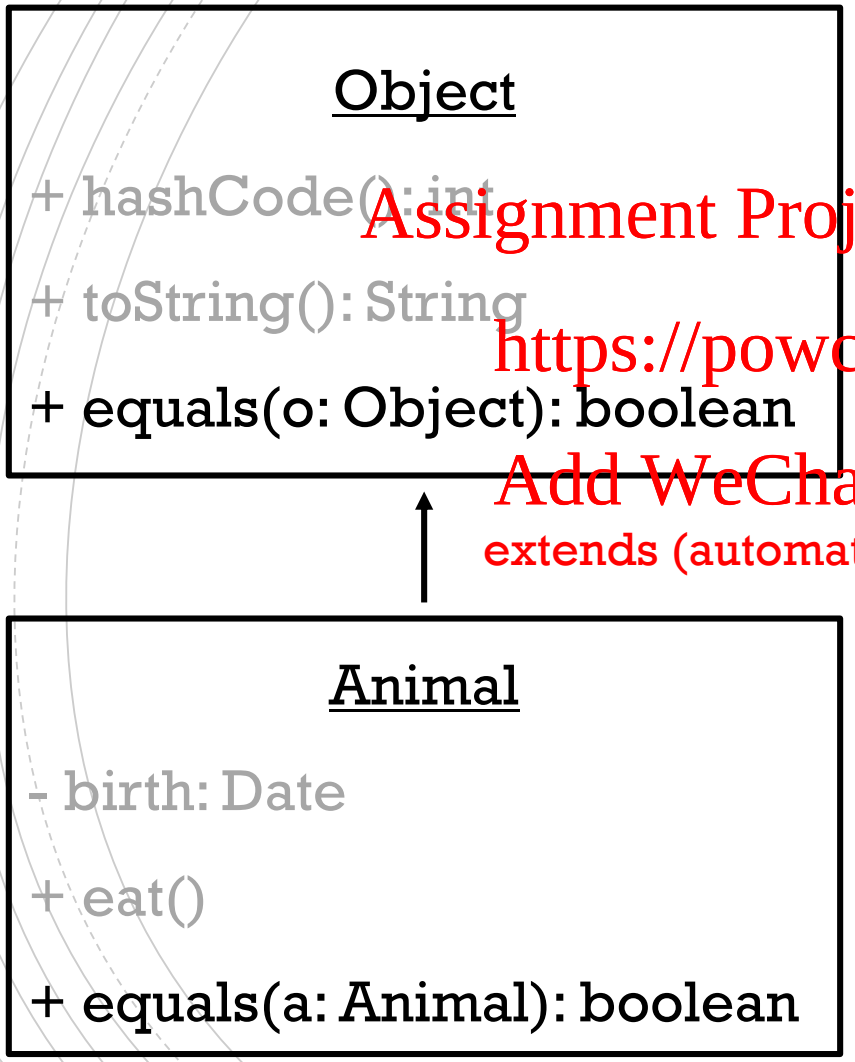
`obj1 == obj2` **has value** `true`

EXAMPLES



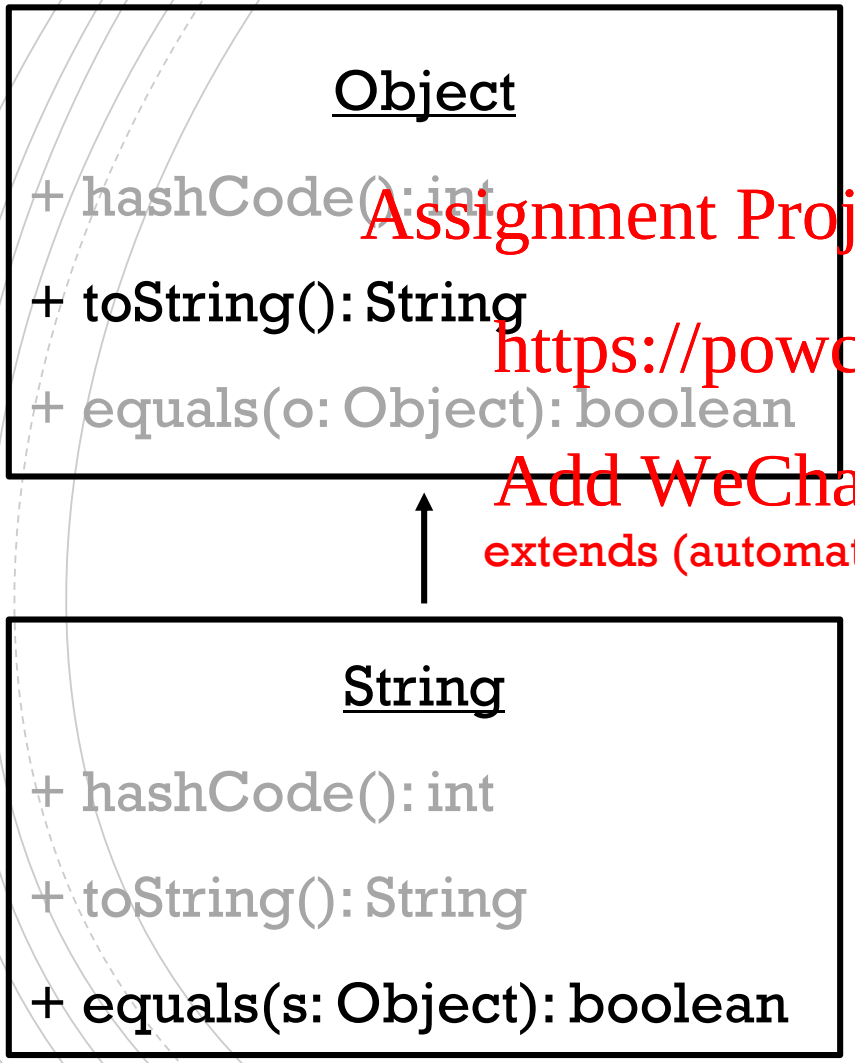
`Animal` **overrides the** `equals()` **method**

EXAMPLES



`Animal` overloads the `equals()` method

EXAMPLES



`String` **overrides the** `equals()` **method**

equals() FROM String

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Overrides:

`equals` in class `Object`

Parameters:

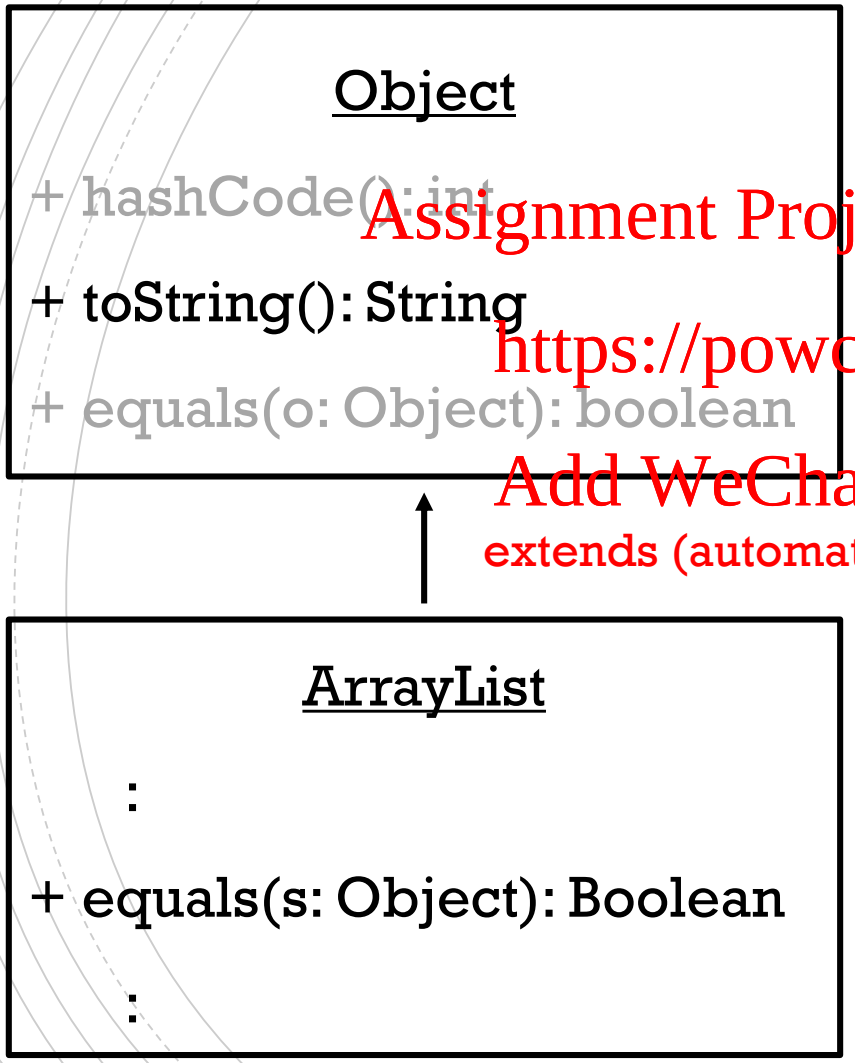
`anObject` - The object to compare this `String` against

Returns:

`true` if the given object represents a `String` equivalent to this string, `false` otherwise

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

EXAMPLES



`ArrayList` **inherits an overridden version of the** `equals()` **method**

Methods inherited from interface java.util.List

`containsAll, equals, hashCode`

https://docs.oracle.com/javase/8/docs/api/java/util/List.html

equals() FROM List

equals

```
boolean equals(Object o)
```

Compares the specified object with this list for equality. Returns `true` if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are *equal*. (Two elements `e1` and `e2` are *equal* if `(e1==null ? e2==null : e1.equals(e2))`.) In other words, two lists are defined to be equal if they contain the same elements in the same order. This definition ensures that the equals method works properly across different implementations of the `List` interface.

Specified by:

`equals in interface Collection<E>`

Overrides:

`equals in class Object`

Parameters:

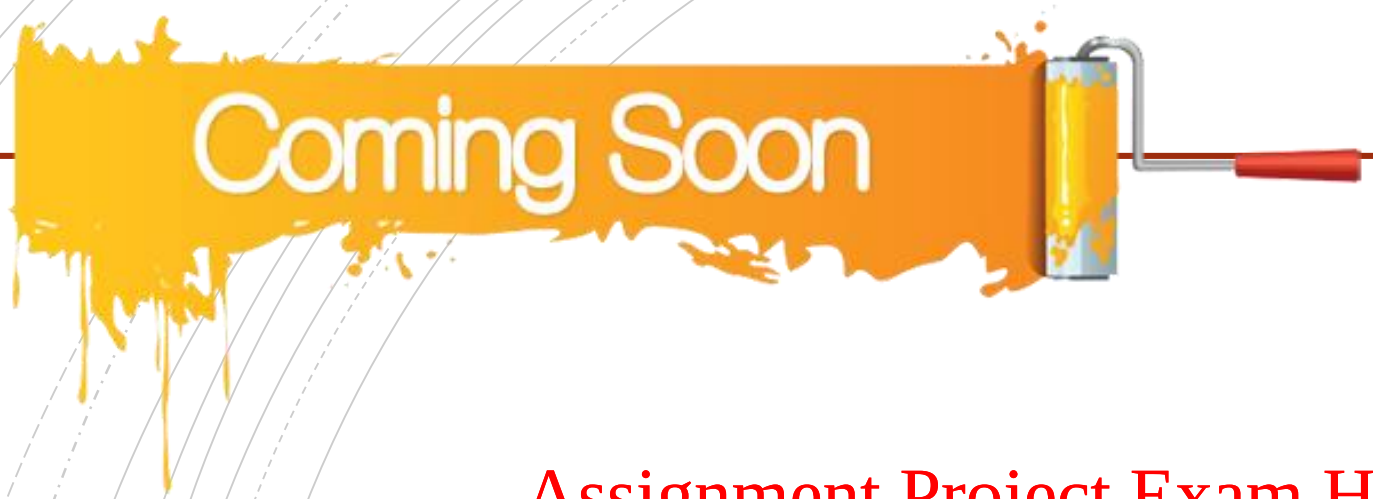
`o - the object to be compared for equality with this list`

Returns:

`true if the specified object is equal to this list`

TO LOOK FORWARD TO

- We will be talking more about interfaces like `List` in a couple of weeks!



In the next video:

- Modifiers and Inheritance
- Type Conversion